MASTER EN SCIENCES INFORMATIQUES

La protection des logiciels contre la malveillance, y compris les virus informatiques

Richoux, Alain

Award date:
1989

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, NAMUR
Institut d'Informatique
Année Académique 1988-1989

LA PROTECTION DES LOGICIELS CONTRE LA MALVEILLANCE, Y COMPRIS LES VIRUS INFORMATIQUES

Mémoire présenté en vue de
l'obtention du grade de
Licencié et Maître en Informatique

par
Alain RICHOUX

Mémoire réalisé
sous la direction de

Monsieur J. RAMAEKERS

Je tiens à remercier ici toutes les personnes sans qui ce mémoire n'aurait jamais vu le jour, et tout particulièrement :

- le personnel du Centre de Technologies Informatiques (CTI) à Fontenay-aux-Roses (France);
- messieurs Labat, Louis et Naciri, membres de l'équipe SAPPHIRE, pour leur accueil chaleureux ainsi que pour leur aide et leur patience de tous les jours;
- monsieur Ferreira, dont la disponibilité et les conseils éclairés m'ont été d'un précieux secours tout au long de mon séjour au CTI.

Ma reconnaissance s'adresse également à monsieur Paul Crismer, qui fut un lecteur aussi patient qu'attentif.

Enfin, je remercie monsieur le Professeur Ramaekers pour avoir accepté de diriger ce mémoire et pour m'avoir procuré un stage aussi passionnant.

AVERTISSEMENT

Une partie de ce mémoire est consacrée aux virus informatiques. Ce sujet étant "délicat", le lecteur attentif pourra remarquer que le nombre de références bibliographiques citées dans cette partie est assez restreint. Il ne s'agit pas d'une conséquence de l'indifférence de l'auteur, mais bien l'aboutissement de la confidentialité des documents scientifiques existants. La communauté scientifique américaine a par exemple jugé le contenu des premiers articles de COHEN un peu trop "risqué"; celui-ci a été instamment prié d'en "neutraliser" les "découvertes" les plus "brûlantes".

Malgré ces petits ennuis, nous avons tenté de donner une vue d'ensemble du phénomène, la moins "journalistique" possible, afin d'aiguillonner la curiosité du lecteur envers un sujet qui nous a passionné.

ERRATUM

- page 3, ne pas relire le premier paragraphe :
  "En examinant ... détournements"

- page 78, note 8 : lire "Quelques attaques ..."
  au lieu de "Les attaques ..."

SOMMAIRE

TABLE DES FIGURES

RESUME

La sécurisation de l'informatique passe par la protection des logiciels contre la malveillance; celle-ci constitue en effet une des facettes les plus importantes du vaste domaine de la sécurité informatique. Parmi toutes les formes qu'elle peut prendre, les virus occupent une place de choix tant est grand leur pouvoir d'infiltration dans les systèmes.

Combattre la malveillance informatique n'est donc pas chose aisée. C'est pourquoi il est préférable d'utiliser pour cela tous les moyens disponibles, tant humains que techniques. Aucune de ces méthodes n'est d'ailleurs infaillible; quelques solutions ont été proposées, qui permettent d'accroître notablement les difficultés rencontrées par les personnes malveillantes, dont font partie les concepteurs de virus informatiques.

ABSTRACT

Information systems security begins with the protection of software against computer crime, which is one of its most important parts. Computer viruses have a dramatic importance among the many pieces of criminal problems because of their great penetration power into computer systems.

Fighting computer crime is therefore not easy. It is thus better to use all the available human or technical methods. None of these is actually fully efficient but some solutions have been proposed. Those can help to notably increase the problems met by persons who have bad intentions, including virus programmers.

INTRODUCTION

Jamais une société humaine n'avait été aussi grande consommatrice d'information que la notre. Pour maîtriser l'accroissement exponentiel du volume des données à traiter, des outils nouveaux ont dû être créés.

Parmi ceux-ci, l'informatique s'est réservée un rôle primordial grâce aux formidables opportunités qu'elle offre pour traiter d'énormes quantités d'information en un temps jamais encore égalé. Le développement de cette technologie aurait ainsi permis la croissance de notre société au cours des dernières décennies.

En conséquence, l'informatisation sans cesse croissante de la société a rendu celle-ci tributaire du bon fonctionnement des ordinateurs. Les intérêts socio-économiques en jeu sont colossaux; il est donc indispensable de renforcer la sécurité informatique à tous les maillons de la chaîne sous peine de voir s'enrayer l'ensemble de ces mécanismes.

Dans ce contexte, nous nous proposons de déployer un éventail, certes limité, des différents risques que présente l'utilisation de l'informatique par des personnes indélicates. Nous développerons ensuite quelques techniques permettant de prévenir ou réparer ces actes de malveillance.

La première partie de ce mémoire est consacrée à la présentation de la malveillance informatique. Nous y montrerons, dans le premier chapitre, l'importance de celle-ci face aux autres risques mettant en péril le bon fonctionnement des systèmes de traitement de l'information. Le deuxième chapitre aura pour but de préciser la malveillance informatique et son impact sur les logiciels. Nous terminerons cette première partie par une présentation des virus informatiques, qui sont en passe de devenir un phénomène de société tant leur développement devient considérable aujourd'hui.

Nous présenterons ensuite quelques pistes permettant de protéger les logiciels contre la malveillance; c'est le sujet de la deuxième partie de ce mémoire.

Des solutions légales, organisationnelles et déontologiques seront d'abord esquissées dans le premier chapitre. Les protections techniques seront ensuite examinées plus en détails, et figureront au chapitre deux. Le dernier chapitre apportera quelques voies de solutions au problème particulier des virus informatiques.

PREMIERE PARTIE

LA MALVEILLANCE

CHAPITRE I

Introduction au problème de la sécurité

CHAPITRE I : INTRODUCTION AU PROBLEME DE LA SECURITE

En guise d'introduction à ce mémoire, nous nous proposons de montrer que la malveillance est l'un des risques les plus graves pour la sécurité informatique en général et celle des logiciels en particulier, et que ce phénomène est en train de s'intensifier. Nous nous inspirerons pour cela de différentes études et statistiques publiées par l'APSAIRD (1).

1. Les types de risques

Les statistiques de l'APSAIRD regroupent les différents risques informatiques en onze classes [APS-86] :

- les risques matériels : il s'agit des "risques matériels accidentels qui ont pour conséquence la destruction au moins partielle de machine, de support ou d'environnement informatiques";
- le vol et le sabotage de matériel;
- les pannes et les dysfonctionnements de matériels et logiciels de base;
- les erreurs de saisie, de transmission et d'utilisation des informations;
- les erreurs d'exploitation : ce sont des oublis, erreurs de manipulation de système, de matériel ou de langage informatique;
- les erreurs de conception et de réalisation;
- la fraude et le sabotage immatériel;
- l'indiscrétion et le détournement d'information;
- le détournement de logiciel : il s'agit ici de la diffusion par un concurrent indélicat de copies ou de dérivés du logiciel piraté;
- la grève et le départ de personnel stratégique;
- risques divers.

---

1) L'APSAIRD (Assemblée Plénière des Sociétés d'Assurances françaises contre l'Incendie et les Risques Divers, 11 rue PILLET-WILL, 75009 PARIS) regroupe des spécialistes de la profession en matière d'informatique.

2. Les types de pertes

Les classes de pertes dégagées sont les suivantes :

- les dommages matériels et annexes : ce sont les coûts de réparation ou de remplacement des matériels informatiques endommagés ou volés;
- les frais supplémentaires : il s'agit des coûts de mise en oeuvre de moyens de secours comme les ressources informatiques de remplacement, les heures supplémentaires, etc;
- les pertes d'exploitation : on classera ici les pertes d'intérêts bancaires, de chiffre d'affaire, de clientèle, etc;
- les pertes de fonds : ces pertes correspondent à le disparition de biens financiers;
- les pertes de biens;
- autres pertes.

3. Estimation des pertes

L'évaluation des pertes dues à l'informatique en 1986 et l'évolution estimée de celles-ci lors de la période 1984-1988 sont présentées respectivement aux tableaux 1 et 2.

Ces estimations tiennent compte du fait que seule une partie des sinistres est connue (certaines études parmi les plus pessimistes vont jusqu'à donner la proportion d'un dixième pour les sinistres connus, ce qui porterait l'estimation des pertes de 1986 à environ 30 milliards de Francs français). En particulier dans le domaine de la fraude, le montant estimé par l'APSAIRD est vraisemblablement largement en-dessous de la réalité. Il semble néanmoins que les tendances dégagées de ces estimations sont assez proches de la réalité.

En examinant le tableau 1 nous pouvons remarquer que ce sont la fraude informatique et le détournement de logiciels qui ont causé le plus de pertes en 1986, immédiatement suivies des risques matériels et des pannes. Les autres risques importants sont les erreurs de saisie, de conception et d'exploitation ainsi que les détournements d'informations.

	Dommages matériels et associés	Frais supplémentaires et pertes d'exploitation	Pertes de fonds et pertes de biens	Autres pertes	TOTAL
Risques matériels	380	650		50	1080
Vol, sabotage de matériel	35	30			65
Pannes		1025			1025
Erreurs de saisie		660	120	20	800
Erreurs d'exploitation		300	20		320
Erreurs de conception		450	100	70	620
Fraude		350	1250	100	1700
Détournement d'informations		220	40	50	310
Détournement de logiciels		1050	150		1200
Grève, départ de personnel		90			90
Divers		60			60
TOTAL	415	4885	1680	290	7270

Tableau 1 : Estimation des pertes dues à l'informatique
(France 1986 - Montants en millions de FF)

En examinant le tableau 1 nous pouvons remarquer que ce sont la fraude informatique et le détournement de logiciels qui ont causé le plus de pertes en 1986, immédiatement suivies des risques matériels et des pannes. Les autres risques importants sont les erreurs de saisie, de conception et d'exploitation ainsi que les détournements d'informations.

Une classification moins fine peut être dégagée en regroupant d'abord les risques accidentels (risques matériels; pannes et dysfonctionnements), ensuite les risques dûs aux erreurs (erreurs de saisie, de transmission et d'utilisation de l'information; erreurs d'exploitation; erreurs de conception et de réalisation) et finalement les risques résultant de la malveillance (vol et sabotage de matériel; fraude et sabotage immatériel; indiscrétions et détournements d'informations; détournement de logiciels). Cette agrégation, présentée à la figure 1.1., nous permet de constater une nette prépondérance des risques dûs à la malveillance (45% du total des pertes) sur les risques

accidentels (30%) et sur les erreurs (24%).

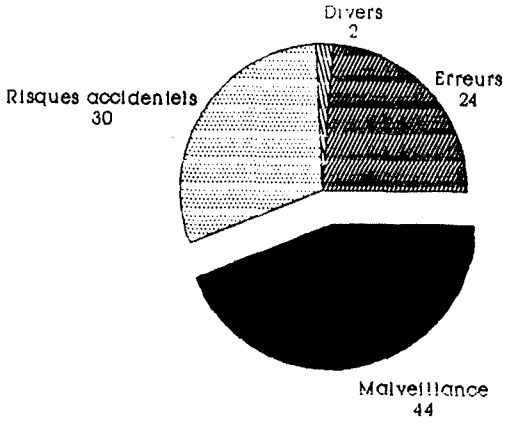

Figure 1.1. : Importance de la malveillance informatique (pourcentage)

De plus, en examinant le tableau 2 qui retrace une estimation des pertes pour la période 1984-1988, nous pouvons constater que la part de

	Dommages matériels et associés	Frais supplémentaires et pertes d'exploitation	Pertes de fonds et pertes de biens	Autres pertes	TOTAL (%)
Risques matériels	15	15	10		+ 15 %
Vol, sabotage de matériel	30	15			+ 22 %
Pannes		10			+ 10 %
Erreurs de saisie		- 10	- 20	- 10	- 11 %
Erreurs d'exploitation		10	- 10		+ 9 %
Erreurs de conception		15	- 15	15	+ 10 %
Fraude		20	30	10	+ 28 %
Détournement d'informations		25			+ 25 %
Détournement de logiciels		20			+ 20 %
Grève, départ de personnel					+ 0 %
Divers					+ 0 %
TOTAL	+ 15 %	+ 5 %	+ 16 %	+ 7 %	+ 8 %

Tableau 2 : Estimation de l'évolution des pertes dues à l'informatique (France 1984-1988 - Ecarts en millions de FF)

la malveillance dans le total des pertes a augmenté au fil des années. D'autres estimations à plus long terme viennent conforter cette tendance [APS-86].

Il semble donc raisonnable d'affirmer que la part de la malveillance dans le coût total des sinistres informatiques est très importante, et que ce phénomène ne fait que s'aggraver depuis quelques années.

4. Les solutions

Nous reportons ici une série de recommandations qui peuvent permettre de réduire les différents risques informatiques [LAM-88]. Certaines de ces recommandations paraîtront ridiculement évidentes tellement elles découlent du bon sens. De nombreux exemples prouvent cependant que ce fameux bon sens n'est pas appliqué partout.

Dans tous les cas, il convient de déterminer quelles seront les protections à offrir au système, compte tenu d'une part du coût de celles-ci et d'autre part de la sécurité qu'elles sont susceptibles de lui apporter.

4.1. L'environnement physique

Il convient d'étudier la salle des machines afin d'éliminer au maximum les risques évitables. En particulier, l'implantation géographique d'un centre informatique dans une zone sujette aux catastrophes naturelles (avalanches, inondations, ...) sera à éviter; de même il sera préférable de ne pas faire passer de canalisation contenant du liquide au-dessus des machines.

4.2. Le contrôle d'accès

Celui-ci est principalement destiné à freiner le malveillance en limitant l'accès aux ressources informatiques (pools de terminaux, d'imprimantes, ordinateur, base de données, ...) au seul personnel autorisé. Le contrôle d'accès permet également de diminuer les pertes dues aux erreurs humaines causées par du personnel non qualifié.

On distingue deux stratégies complémentaires pour interdire l'accès aux ressources aux personnes non autorisées. Ce sont le contrôle d'accès physique et le contrôle d'accès logique.

4.2.1. Le contrôle d'accès physique

Il s'agit de protéger physiquement l'accès aux salles qui abritent les ressources informatiques physiques (terminaux, ordinateur, imprimantes, ...). L'entrée de ces salles sera refusée aux personnes jugées non habilitées à utiliser le matériel qu'elles renferment.

Différents moyens peuvent être utilisés pour réaliser un tel contrôle. Le port du badge est, semble-t-il, la solution la plus classique; elle est efficace à condition qu'une vérification systématique soit réalisée. Une solution plus automatisée consiste dans l'utilisation d'un sas d'entrée équipé d'un lecteur; celui-ci peut être du type clavier où l'on entre un code d'accès, du type lecteur de carte magnétiques ou de carte à microprocesseur, ou encore lecteur de reconnaissance biométrique (2).

La solution choisie sera fonction de son coût face au niveau de sécurité qu'elle permet d'atteindre. Pour des applications demandant un niveau de sécurité élevé, une combinaison des techniques précédentes donnera probablement satisfaction aux responsables; lorsque par contre le coût est une variable décisive, il se peut que la solution de protection choisie ignore totalement le contrôle d'accès physique pour ne plus reposer que sur la protection offerte par les barrières logiques.

4.2.2. Le contrôle d'accès logique

Il s'agit ici de vérifier automatiquement qu'une personne désireuse d'accéder à une ressource en a bien le droit. Ce contrôle est généralement réalisé lors de la demande d'accès à cette ressource (physique, comme un serveur, ou logique, comme un fichier).

La réalisation de ce type de contrôle peut aller de la simple demande d'un mot de passe donnant accès au système jusqu'à un protocole d'authentification basé sur des technologies avancées (carte à microprocesseur).

---

2) Les techniques présentées dans cette section seront étudiées plus en détail par la suite.

4.3. L'entretien du matériel

L'entretien du matériel est une condition quasi nécessaire pour la diminution du risque informatique. Les contrôles seront pratiqués de manière systématique et suivant une planification.

4.4. Les contrats

Les contrats permettant de se protéger en limitant l'incidence des pertes sont les contrats d'assurances; certaines clauses des contrats de vente ou d'entretien peuvent également servir ce but.

4.4.1. Les contrats d'assurances

Ils couvrent généralement les pertes matérielles (physiques), rarement les pertes d'exploitation et exceptionnellement les pertes de données. La raison réside probablement dans la difficulté d'estimer de telles pertes.

4.4.2. Les contrats de vente ou d'entretien

Ils peuvent comprendre des clauses prévoyant des pénalités de retard de livraison ou de remise en état, ce qui aide également à diminuer les pertes subies lors d'un sinistre.

4.5. La redondance des moyens

Une solution pour limiter les incidences des pannes de matériel informatique (et donc de certaines pertes) est de prévoir des moyens de report de certaines applications importantes sur d'autres machines. La configuration "centralisée", où un gros ordinateur traite tous les problèmes de l'organisation, n'est donc pas la plus judicieuse dans cette optique. Le choix de plusieurs ordinateurs plus petits, réalisant la même fonction globale et dont chacun pourrait accueillir les applications vitales de l'organisation, semble plus adéquat.

On peut également noter ici que le choix d'un ordinateur à tolérance

aux pannes pourra dans certains cas permettre la continuité du traitement, par exemple en mode dégradé, là où un ordinateur classique aurait dû être stoppé. Le fonctionnement de ce type de machines est généralement basé sur le même principe de redondance des moyens (stratégie de vote au sein d'un nombre impair de processeurs, par exemple).

4.6. Les centres de back-up

Il s'agit d'une solution mixte entre la protection par contrat et la redondance des moyens.

Le principe est simple : une solution pour se protéger contre les risques informatiques est de doubler entièrement le centre informatique et de faire fonctionner le second centre en cas de panne du premier. Ce scénario très coûteux est utilisé dans certaines organisations où la sécurité doit être maximale (3).

Le centre de back-up est dérivé de cette solution : il représente des énergies informatiques de réserve assujetties à la signature d'un contrat. Il constitue ainsi pour l'organisation une assurance de la continuité de ses applications vitales. Moyennant le paiement d'une cotisation de réservation (abonnement que l'on peut assimiler à une assurance) et d'une taxe forfaitaire (que l'on peut classer dans la catégorie des frais supplémentaires), l'organisation peut ainsi utiliser les moyens informatiques que le centre de back-up met à la disposition de ses abonnés en cas de sinistre.

4.7. La protection des informations

La perte, la divulgation ou la modification de certaines informations d'une organisation peut représenter une perte industrielle (un brevet ou un secret de fabrication, par exemple) ou même une perte commerciale (comme le vol du fichier des clients). De plus, nous avons vu que peu de compagnies d'assurances acceptent de couvrir ces risques. Il convient donc pour une organisation moderne de protéger ce capital.

Les protections à mettre en oeuvre pour se protéger des pertes d'information sont bien connues : il s'agit d'effectuer des copies de sauvegarde et

---

3) Par exemple dans les centrales nucléaires, où le centre informatique est parfois doublé ou même triplé.

de les disséminer de manière à ne pas les perdre toutes en cas de sinistre.

Les moyens de protection contre la modification illicite ou contre la divulgation seront examinées par la suite. En fonction de ces techniques, une politique de protection des informations pourra être définie selon l'importance attribuée à chacune des informations.

5. Conclusion

Nous avons exposé dans ce chapitre une estimation des pertes en fonction des différents types de risques. Il est ainsi apparu que la part des risques dûs aux malveillances est très importante et que ce phénomène ne fait que s'amplifier.

Nous avons ensuite proposé une série de recommandations permettant de diminuer les risques informatiques ou d'en limiter les conséquences pour les organisations. Certaines de ces recommandations sont axées sur d'autres facteurs de risques que la malveillance; il convient en effet de réduire l'ensemble des risques si l'on veut améliorer la sécurité de manière appréciable. La lutte contre la malveillance ou la sécurité des logiciels ne constituent qu'une facette d'un vaste problème : *la sécurité informatique.*

CHAPITRE II

La malveillance

CHAPITRE II : LA MALVEILLANCE INFORMATIQUE

L'analyse du chapitre précédent a mis en évidence l'importance de la lutte contre la malveillance dans les systèmes informatiques actuels. Il semble également que cette forme de délinquance risque de prendre beaucoup d'ampleur dans les années à venir si l'on ne s'attache pas à trouver des solutions permettant de la combattre efficacement. Une présentation de différents types de protection fera l'objet de la seconde partie de ce mémoire

Quant au présent chapitre, il a pour but de préciser la notion de malveillance ainsi que ses effets sur les systèmes informatiques, sur les données et sur les logiciels. Une section sera spécialement consacrée à l'étude des virus informatiques; ceux-ci constituent en effet une classe de malveillance informatique particulièrement pernicieuse, qui semble de plus en plus utilisée ces dernières années.

1. La malveillance informatique : définition

La malveillance informatique est définie comme un

> *acte intentionnel impliquant une technologie informatique qui a (ou aurait pu) soit faire subir une perte à une victime, soit faire bénéficier son auteur d'un profit illicite, soit les deux* [PAR-76].

On peut distinguer trois classes de malveillance, selon le type de ressource informatique qu'elles visent. Ces trois types de "cibles" de la malveillance sont les systèmes, les données, et les programmes (ou logiciels).

2. La malveillance contre les systèmes

Ce type de malveillance regroupe les pénétrations illicites d'un système informatique ou d'une partie de celui-ci, ainsi que les dégradations volontaires des performances de ce système. Nous en donnons ici un rapide aperçu, qui sera suivi de quelques pistes de réflexion permettant de contrer

les malveillances de ce type (1).

Les attaques violentes contre les systèmes (vol et sabotage de matériel) sont également classées ici. Nous n'en parlerons pas dans la suite car il s'agit d'un problème connu bien avant l'apparition de l'informatique.

2.1. Définition d'un système informatique

Précisons tout d'abord ce que nous entendons par système informatique : *Un système informatique est un ensemble de matériels, de logiciels, de données et, le cas échéant, de personnes. Il est organisé de manière à accomplir un ensemble de fonctions déterminées de traitement de l'information* [GIL-88].

Nous utiliserons souvent la contraction "système" pour "système informatique".

2.2. Les chevaux de Troie

Le principe de la stratégie mise en oeuvre pour pénétrer illicitement un système est connu depuis l'Antiquité : lorsque les Grecs levèrent le siège de la ville de Troie, abandonnant un gigantesque cheval de bois, les Troyens crurent à un hommage rendu à leur courage; ils installèrent donc cette statue dans leurs murs avant de fêter leur victoire. Mal leur en prit car Ulysse et ses soldats étaient cachés à l'intérieur; ils sortirent une fois la nuit venue et la ville fut mise à sac (2).

Une telle stratégie appliquée à l'informatique mène à écrire un programme qui réalise un but que l'on s'est fixé ("mettre a ville à sac"-pénétrer le système) tout en ayant l'air attrayant ou inoffensif ("une statue de bois" - un jeu, un programme utilitaire, ...). L'exemple suivant existe probablement depuis que l'accès aux ordinateurs est subordonné à l'introduction d'un mot de passe.

---

1) Le lecteur désireux d'approfondir ce sujet, dont l'analyse sort du cadre de ce travail, peut par exemple se référer aux recherches de MM. PAANS, HERSCHBERG et HOBOKEN (notamment : [HER-84], [HER-87] et [HOB-84]).

2) VIRGILE, *L'Enéide*, Livre 2.

Le cheval de Troie est ici un programme qui reproduit l'écran de "LOGIN" envoyé par le système d'exploitation. Un utilisateur désireux d'accéder à la machine lui fournira son nom accompagné de son mot de passe, en croyant qu'il a affaire au module spécialisé du système d'exploitation. Le nom et le mot de passe seront alors recopiés dans une zone appartenant au propriétaire du cheval de Troie, puis le programme affichera un message signalant que le mot de passe était incorrect et terminera la session. L'utilisateur accédera à l'ordinateur au second essai (son interlocuteur sera alors le "vrai" programme de LOGIN) en croyant avoir fait une erreur dans la frappe de son mot de passe lors de la première tentative.

Le propriétaire du cheval de Troie est dès lors en possession du nom et du mot de passe de cet utilisateur.

Beaucoup de variantes de ce principe ont été imaginées. La plus simple est peut-être d'intituler le programme "malveillant" d'un nom attrayant, comme *game*, *enjoy* ou même ... *sex* (3) !

Un cheval de Troie remarquablement pernicieux a été décrit par Ken THOMPSON (4) [THO-84].

Le compilateur du langage C est altéré de la manière suivante :

1. Il est capable de détecter si le programme source qu'on lui demande de compiler est celui de la commande "LOGIN" de UNIX (le système d'exploitation UNIX est presque entièrement écrit en C). Si c'est le cas, le compilateur ajoute un "bug" au résultat, de telle sorte que la commande LOGIN acceptera soit le mot de passe correct d'un utilisateur, soit un mot de passe particulier connu de THOMPSON.
2. Il est également capable de détecter si le source qu'on lui fournit est le sien. Dans ce cas, il ajoute deux "bugs" au résultat : les deux altérations qui viennent d'être citées (détection du LOGIN et de lui-même).

Le cheval de Troie se perpétue ainsi, bien qu'aucune trace de son code n'apparaisse plus nulle part dans les listings source.

Ce cheval de Troie peut être éliminé de deux manières : soit en vérifiant le code du compilateur C pour en retirer les lignes indésirées (à

---

3) Un programme appelé *Mac Sex* simule un formatage du disque dur des MacIntosh, au moment psychologique ... douche froide garantie ! (Notons que, dans ce cas précis, le programme est totalement inoffensif).

4) THOMPSON est l'un des créateurs de UNIX.

l'aide d'un décompilateur dont on est sûr qu'il est "intègre" ...), soit en recompilant son source à l'aide d'un autre compilateur, intègre celui-là.

Mais la morale, comme le dit THOMPSON, est évidente : *on ne peut pas faire confiance à du code qu'on n'a pas entièrement créé soi-même.*

2.3. Les attaques contre les mots de passe

Cette stratégie de pénétration peut avoir lieu lorsque le système visé est protégé par un contrôle d'accès basé sur l'utilisation de mots de passe. Une personne malveillante peut alors tenter de deviner le mot de passe d'un utilisateur patenté et de pénétrer le système.

A première vue, une telle manière de procéder tient de la gageure. En effet, si l'on suppose une moyenne de cinq caractères par mot de passe, il y a plus de trente milliards de possibilités de codage si 125 caractères ASCII sont utilisés (5).

Ce nombre est cependant réduit à neuf cent seize millions (trente trois fois moins) si l'on n'utilise que les cinquante-deux lettres (majuscules et minuscules) et les dix chiffres; il n'est plus que d'environ douze millions de possibilités si seules les lettres minuscules sont utilisées. La probabilité de découvrir un tel mot de passe "au hasard" est malgré tout négligeable.

Mais, malheureusement pour la sécurité des systèmes, il faut encore relativiser ces nombres. En effet, peu de gens aiment ou savent retenir des chaînes de caractères sans signification [HOB-84]. Une proportion non négligeable des mots possibles ne sera ainsi "jamais" utilisée dans la réalité (6).

Il semble donc qu'en moyenne, si seuls des mots qui ont du sens sont utilisés comme mots de passe et si leur longueur moyenne est de cinq caractères, alors un attaquant aura une chance sur cinq cent mille de trouver le bon mot de passe au premier essai.

---

5) $125^5 = 30\ 517\ 578\ 125$

6) Ainsi, le Petit Larousse 1987 définit cent mille substantifs et nonante mille noms propres. En y ajoutant les pluriels, les féminins, les verbes conjugués et les noms propres que Larousse ne connaît pas, on arrive à un total probablement en-dessous du million ! On peut en conclure que nonante pour-cent des chaînes de caractères ne seront jamais utilisées dans la pratique ...

En poursuivant le même raisonnement, on remarque qu'un utilisateur choisira son mot de passe de telle sorte qu'il puisse s'en souvenir facilement et le retrouver aisément en cas d'oubli. Or la mémoire humaine est ainsi faite qu'il est plus facile de se souvenir de la proposition : *mon mot de passe, c'est le prénom de mon frère,* plutôt que de : *mon mot de passe, c'est le mot "voiture".* C'est ainsi qu'un attaquant rusé peut être guidé dans sa recherche s'il connaît un peu la personne qu'il attaque [HER-87]. Il essayera en priorité les mots de passe "triviaux" comme :

- le nom d'un objet qui est sous les yeux de la personne assise au terminal (écran, clavier, ...);
- le nom de l'utilisateur, son prénom, son numéro de groupe, de compte, ...;
- un élément de la vie privée de l'utilisateur (numéro d'immatriculation, prénom d'un proche, ...);
- un mot qui vient tout de suit à l'esprit d'une personne qui cherche un mot de passe (secret, 007, ...) (7).

Un autre moyen de ne pas oublier son mot de passe est tout simplement de le noter quelque part. L'endroit choisi comme aide-mémoire est parfois surprenant de naïveté : il arrive fréquemment que le mot de passe soit écrit directement sur le terminal, ou dans un manuel, pratiquement sous les yeux d'une personne curieuse ou mal intentionnée.

En résumé, le premier problème rencontré dans l'utilisation des mots de passe est d'ordre humain. Un second problème, plus technique celui-là, va à présent être abordé.

La plupart du temps, le mot de passe conditionnant l'accès à une ressource doit être composé sur un clavier. Bien qu'en général l'écho des caractères frappés n'apparaît pas à l'écran, il est souvent possible de déterminer le mot de passe en observant les touches enfoncées par l'utilisateur légitime [ANO-88]. Si par exemple l'attaquant a découvert quatre possibilités de frappe pour chacun des caractères du mot de passe (8) alors

---

7) Notons qu'il existe des dictionnaires spécialisés à l'usage des pirates, reprenant les mots qui reviennent le plus dans le choix d'un mot de passe.

8) Par exemple, il a pu voir que le premier caractère se trouve "autour de" Q; il retient les possibilités A, Q, 1 et W dont les touches sont près du Q sur un clavier QWERTY.

celui-ci se trouve dans l'ensemble des 1024 chaînes possibles. En combinant cette attaque avec la précédente, il reste statistiquement dans la course un peu plus de 120 mots qui ont du sens, parmi lesquels il sera facile de découvrir la solution si l'on connaît un peu l'utilisateur visé. Nous sommes donc loin des trente milliards de possibilités du départ !

De plus, une recherche exhaustive par programme est souvent envisageable, même dans le cas où l'attaquant ne dispose d'aucun renseignement pour guider sa recherche.

Une solution technique simple permettant de gêner considérablement ces attaques est d'implanter un compteur d'essais de connections infructueux. On peut alors imaginer plusieurs stratégies de protection, parmi lesquelles : imposer un délai relativement long, après N essais infructueux, avant d'autoriser la N+1ème tentative de connexion au système; imposer un changement fréquent des mots de passe, afin d'éviter qu'un intrus ne puisse conserver l'accès au système pendant une longue période; éduquer les utilisateurs pour éviter les "aide-mémoires" et les mots de passe triviaux; etc.

2.4. Attaques exploitant les faiblesses du système

Ce troisième type d'attaque peut avoir lieu lorsqu'une personne travaillant sur le système juge son champ d'action trop restreint et commence à tenter d'élargir quelque peu son espace vital. Ceci peut inclure le désir de lire ou modifier certaines données confidentielles, d'accélérer son propre travail en augmentant son niveau de priorité, ou de travailler gratuitement, par exemple [HOB-84].

Le système de sécurité doit être conçu de manière à empêcher ces attaques. Il s'agit ainsi d'interdire toute modification non autorisée des attributs d'une ressource afin d'éviter qu'un utilisateur ne puisse obtenir de manière illégale le droit d'y accéder. En particulier, le fichier des mots de passe doit être illisible pour tout utilisateur. De même, il est indispensable de prendre des mesures pour confiner efficacement un utilisateur au sein de son niveau de sécurité, afin d'éviter qu'il ne fasse augmenter celui-ci pour bénéficier de droits plus élevés.

Ces tâches sont souvent dévolues au système d'exploitation. Mais il semble que bien peu d'entre eux soient sûrs à ce niveau et qu'une sécurité optimale ne peut être atteinte que grâce à des modules matériels spécialisés [HER-84].

2.5. Les dégradations du service

Cette attaque consiste à rendre le système inutilisable (ou à en diminuer les performances) pour tout ou partie de l'ensemble des utilisateurs. Ceci peut par exemple être réalisé en saturant certains périphériques pour rendre leur usage impossible par d'autres personnes en utilisant un programme d'entrée/sortie qui boucle (pour plus de détails, voir [HER-84] p. 267), ou encore en lançant un grand nombre de travaux inutiles mais gros consommateurs de temps CPU pour dégrader les performances du système. Remarquons que ces attaques peuvent être réalisées aux frais d'un tiers dont on a pu découvrir le mot de passe ...

2.6. Les attaques contre les réseaux

Les réseaux constituent plutôt un ensemble de systèmes informatiques qu'un système proprement dit. Nous définirons en effet un réseau comme un

système de communication permettant le dialogue en temps réel entre plusieurs systèmes informatiques.

Nous ne ferons pas ici la traditionnelle distinction entre les réseaux locaux et les réseaux publics car les problèmes rencontrés dans ces deux types d'équipements ne sont pas fondamentalement différents.

Les réseaux prennent de plus en plus d'importance dans notre société. Les organisations gouvernementales et surtout intergouvernementales se basent dans une mesure importante sur l'existence de moyens de transmission rapides et fiables; il en est de même dans le monde des affaires, pour ne citer que ceux-là [MAD-88].

Ce "mégaréseau" mondial est ainsi devenu une cible très tentante pour le terrorisme international : la destruction de stations clés pour le guidage de satellites ou de câbles sous-marins pourrait avoir des conséquences graves et rapides en cas de crise internationale, par exemple. Certaines attaques violentes contre des centres informatiques ont d'ailleurs déjà eu lieu en Europe, de la part de groupuscules politiques comme Action Directe en France (attaque des installations informatiques chez PHILIPS en 1980), ou même dues à des organismes apolitiques comme le CLODO (quelques attaques dans le sud de la France au début des années 80).

Ces attaques posent le problème souvent négligé de la sécurité physique des installations informatiques civiles. Les solutions se basent d'une part sur la protection physique des sites, et d'autre part sur des plans de restauration rapide et efficace des transactions les plus prioritaires. Dans cette optique, la décentralisation des serveurs et une certaine redondance des moyens pourraient constituer un choix judicieux.

3. La malveillance contre les données

La malveillance contre les données contenues dans un système informatique peut prendre deux formes différentes, auxquelles correspondent deux types de protection possibles.

On peut ainsi désirer rester propriétaire de certaines données, en empêchant les tiers d'y accéder. Nous qualifierons ces données de *confidentielles*.

Une autre classe de protection concerne les modifications : on veut être certain que les données protégées ne peuvent être modifiées que par l'autorité habilitée à le faire. On désire ici s'assurer de l'*intégrité* de ces données.

Il est à remarquer que l'utilisation de réseaux pour transmettre des données confidentielles ou nécessitant de rester intègres pose un problème difficile du fait du caractère public inhérent aux réseaux. C'est pourquoi nous examinerons brièvement les problèmes spécifiques introduits par l'utilisation des réseaux, après avoir présenté les deux types de malveillance dégagés ci-dessus.

3.1. Malveillance passive

La malveillance passive est directement dirigée contre le caractère confidentiel des données; il s'agit de la lecture illicite d'informations.

Une organisation peut en effet être amenée à conserver des informations sensibles dont elle désire garder l'exclusivité. Citons simplement les brevets ou les secrets de fabrication, le fichier des clients, les données auxquelles on ne peut accéder que moyennant finances (base de données), ou encore des données concernant les tiers comme les archives d'un hôpital (à ce sujet, voir l'article d'Isabelle GRANDJEAN [GRA-86] qui analyse le problème de l'informatisation des données médicales).

Les solutions à ce problème revêtent deux aspects complémentaires. On peut tout d'abord tenter interdire la lecture de ces données par une méthode de contrôle d'accès ou d'authentification. On peut ensuite agir sur ces données de manière à ce qu'une lecture non autorisée de celles-ci n'apprenne rien à l'indiscret au sujet du contenu de ces données.

Une première technique utilisable est la cryptographie, qui transfère le secret des données sur le secret d'une clé. Une seconde technique possible est l'utilisation d'un schéma à seuil [CAM-87] basé sur la fragmentation-dissémination des données à protéger [RAN-87], qui oblige l'attaquant à pénétrer un minimum de N centres de stockage protégés avant de pouvoir attribuer une signification correcte aux données qu'il a surprises.

Ces méthodes (cryptographie, contrôle d'accès, schéma à seuil) seront détaillées dans le chapitre consacré aux protections techniques des logiciels. Remarquons qu'il peut paraître paradoxal d'appliquer aux données les protections habituellement dévolues aux logiciels, et vice-versa. La raison est qu'il est pratiquement impossible de définir une frontière nette entre données et programmes, puisque ceux-ci ne sont finalement que de l'information; c'est au moment où cette information sera interprétée que l'on pourra établir une distinction. Notons de plus qu'une telle distinction n'est pas immuable; ainsi, tout "programme" exécutable ne pourra-t-il s'exécuter qu'après avoir été *chargé* par un *loader*, et donc lui avoir servi de "donnée" ! De même, il est possible pour chaque quantité d'information de construire un programme qui fait effectuer un certain traitement à la machine seulement dans le cas où cette information interprétée est équivalente à une autre quantité d'information prédéfinie ... ce qui revient à construire un "interpréteur" pour un nouveau langage, et donc de considérer l'information interprétée comme un programme !

En résumé, il ne nous semble pas pertinent d'établir une distinction tranchée entre confidentialité des données et confidentialité des logiciels, car les protections envisagées ne dépendent que du mode de stockage et de distribution de l'information et pas de la nature de l'interprétation dont elle fera plus tard l'objet. Il en sera de même pour l'intégrité de l'information. Nous parlerons donc d'information plutôt que de données ou de logiciels.

3.2. La malveillance active

Ce type de malveillance concerne les modifications non autorisées d'information; il s'agit donc d'atteintes à l'intégrité de l'information.

Les conséquences de modifications de certaines données sensibles peuvent être graves et coûteuses pour l'organisation qui en est propriétaire. Nous citerons ici à nouveau l'exemple des données médicales. On peut également imaginer ce qu'il adviendrait de la crédibilité d'un système de consultation de base de données si ses données ou ses programmes subissaient des modifications anarchiques ! Il existe ainsi dans chaque système de l'information qui doit être modifiable, mais seulement par certaines personnes dans certaines circonstances.

Il faut dès lors prévoir une méthode permettant d'interdire toute modification non autorisée, ou du moins mettre en oeuvre une technique permettant la détection des modifications non autorisées.

Une interdiction pure et simple des modifications illicites peut être réalisée par contrôle des droits d'accès logiques ou physiques. La vérification de l'intégrité, quant à elle, fera appel à des techniques de détection cryptographiques comme l'authentification ou la signature numérique (voir le chapitre traitant de la protection des logiciels).

3.3. Les attaques en cours de transmission des données

La nature publique des réseaux rend ceux-ci plus vulnérables aux attaques tant passives qu'actives. Surprendre une communication est à la portée de n'importe quel bricoleur; de même, il n'est guère plus compliqué de se brancher "en coupure" entre deux correspondants de manière à effectuer en temps réel un filtrage de leurs communications.

Les solutions à ces problèmes sont très semblables à celles qui sont utilisées dans un environnement fermé (chiffrement, ...); elles seront présentées dans la seconde partie.

Mais d'autres problèmes apparaissent lors d'une transmission de données. Les interlocuteurs peuvent désirer conserver secrète l'existence même de la communication. Chacun veut être sûr de l'identité de son correspondant, et avoir l'assurance que celui-ci ne pourra pas nier avoir envoyé un

message ou inversement l'avoir reçu; on désire également ne pas être accusé d'avoir envoyé un message si on ne l'a pas fait.

Les techniques existantes de protection peuvent se répartir en huit classes [KRA-87] (9) :

1. Le *bourrage* : en cas d'absence d'utilisation de la liaison, on émet des messages inutiles, destinés à empêcher un espion de savoir à quel moment ont lieu les véritables communications.
2. Le *chiffrement* : son but est d'empêcher qu'un espion ne puisse comprendre la signification des données transmises. On protège donc ici la confidentialité du message.
3. Le *contrôle d'accès* : il s'agit de n'autoriser l'accès aux ressources du réseau qu'à un ensemble bien défini d'usagers.
4. Le *contrôle de routage* : lorsque c'est possible, on peut choisir la route que suivront les données (par exemple pour contourner une partie peu sûre du réseau).
5. L'*échange d'authentification* : il consiste en protocole spécial basé sur des techniques parfois très élaborées (comme la "zero knowledge", qui permet de prouver que l'on détient un secret sans rien en révéler [FIA-87, GOL-86, WAY-87] et à l'issue duquel chacun des deux interlocuteurs est convaincu de l'identité de l'autre).
6. La *notarisation* : elle est effectuée auprès d'une troisième entité dont l'impartialité ne peut être mise en doute. Les deux correspondants ne pourront pas nier avoir émis ou reçu les messages qui l'ont effectivement été, de même qu'on ne pourra pas leur opposer de faux.
7. La *signature* : basée sur des méthodes cryptographiques, elle a les mêmes propriétés qu'une signature manuelle classique : le message signé ne peut être modifié que par son signataire, ce qui donne à celui-ci certaines garanties; de même, le destinataire est en mesure de prouver que c'est son correspondant qui est l'auteur du message qu'il tente de lui opposer.
8. La *vérification d'intégrité* : elle permet de détecter s'il y a eu une altération du message lors de sa transmission. Certaines techniques sont dirigées spécifiquement contre la malveillance (méthodes cryptographiques), d'autres ont pour but de détecter et éventuellement de

---

9) Le classement présenté est purement arbitraire; il ne reflète aucune hiérarchie éventuelle dans l'efficacité de ces méthodes.

corriger des erreurs lors de la transmission (codes détecteurs et correcteurs d'erreurs).

Certaines de ces méthodes sont également utilisées dans un environnement fermé; il s'agit du chiffrement, du contrôle d'accès, de la vérification d'intégrité, et, dans une moindre mesure, de la signature. Nous reparlerons de ces techniques dans le chapitre consacré aux protections des logiciels.

Quant aux autres (l'échange d'authentification, la notarisation, le bourrage et le contrôle de routage) elles sont plus spécifiques à un environnement réseau. Comme la sécurité des réseaux n'entre pas dans le cadre de ce mémoire, le lecteur intéressé devra se référer à des ouvrages plus spécialisés, comme par exemple [KRA-87] ou [DAV-84].

4. La malveillance contre les logiciels

Nous entendrons, dans ce chapitre, un logiciel comme une

quantité d'information dont la finalité est d'être interprétée comme un programme.

Il ne s'agit pas d'une définition rigoureuse, mais elle suffit à notre propos.

Nous distinguons trois types de malveillance dirigée spécifiquement contre les logiciels :

- les recopies illicites,
- les exécutions non autorisées,
- les modifications interdites.

4.1. Les recopies illicites

La création d'un logiciel représente un investissement important qu'il convient de rentabiliser en en commercialisant suffisamment d'exemplaires. La multiplication des copies non autorisées diminue les revenus légitimes des concepteurs pour qui seules les ventes constituent une source de recette.

Notons ici que ce type de malveillance regroupe toutes les copies non autorisées de logiciels, qu'elles aient été réalisées dans un but gratuit ou onéreux.

4.2. Les exécutions non autorisées

Ce type de malveillance peut découler du précédent : l'exécution d'une copie illicite d'un logiciel est elle-même illicite. On peut également désirer que la version originale du logiciel ne puisse être exécutée de plein droit que par certaines personnes (par exemple, les commandes de gestion du système sur une machine partagée).

4.3. Les altérations

Une altération est définie comme une *modification non autorisée;* il s'agit donc d'une atteinte à l'intégrité du logiciel. Ce type de malveillance peut avoir pour but le plagiat (revente sous un autre nom d'un logiciel légèrement modifié) ou de la fraude (modification d'un programme de transferts de fonds, par exemple). L'inclusion de bombes logiques ou de chevaux de Troie dans un programme sera également classée ici, ainsi que les modifications dues aux virus informatiques évoquées au chapitre suivant.

5. Conclusion

Après avoir défini la malveillance informatique, nous en avons détaillé différentes facettes. Les chevaux de Troie, les attaques contre les mots de passe ou exploitant les faiblesses du système, les dégradations volontaires du service et la sécurité des réseaux constituaient la première classe de malveillance : celle qui est dirigée contre les systèmes.

Le second type de malveillance envisagé regroupait les attaques passives ou actives contre l'information. Le cas particulier des informations transmises par réseau a également été évoqué.

La malveillance dirigée spécifiquement contre les logiciels constituait la troisième classe. On y rencontrait les recopies illicites, exécutions non autorisées et autres altérations illicites.

Nous allons à présent examiner plus en détail une forme un peu particulière d'altération des logiciels : les virus informatiques. C'est le sujet du troisième chapitre.

CHAPITRE III

Les virus informatiques

CHAPITRE III : LES VIRUS INFORMATIQUES

Les virus informatiques constituent une menace particulièrement grave pour la sécurité des systèmes en général, et des logiciels en particulier. C'est pourquoi nous allons nous y attarder quelque peu.

Nous examinerons ici le déroulement d'une attaque virale. La propriété d'infection fera ensuite l'objet d'une attention toute particulière car elle suffit, comme nous pourrons le constater, à définir un virus. Nous terminerons en précisant quelque peu la notion de déclenchement.

1. Définition et commentaires

1.1. Définition

Un virus informatique est défini comme

un programme pouvant infecter d'autres programmes en y incluant une version éventuellement modifiée de lui-même. [COH-87]

Pour illustrer cette notion, nous présentons l'algorithme d'un virus à la figure 3.1. (le pseudo-langage utilisé est défini en annexe, page 129).

```
REPETER
      fich <- choisir_un_programme_au_hasard;
JUSQU'A (pas_encore_infecte (fich));
ajout_virus_au_fichier (fich);
aller_a (debut_programme_infecté);
```

Figure 3.1. : algorithme d'un virus simple

1.2. Commentaires

Cette définition d'un virus peut inspirer certaines réflexions. Remarquons tout d'abord que, comme nous l'avions annoncé, le concept clé est

l'infection, concept qui sera analysé plus loin. Remarquons ensuite qu'il n'est pas nécessaire qu'un programme "infectieux" nuise au système qu'il infecte pour être appelé virus, seule compte la propriété d'infection. Des virus "utiles" ont ainsi été écrits, dont le but était par exemple de leurrer un autre virus, nuisible celui-là, de telle sorte qu'un programme infecté par le "bon" virus ne soit jamais réinfecté par le "mauvais" virus; ou encore de réaliser des compressions de fichiers afin d'économiser de la place sur la mémoire auxiliaire [COH-87]. En troisième lieu, remarquons qu'un virus peut se modifier de génération en génération. Cette caractéristique importante de la définition sera examinée plus loin.

Une dernière remarque peut être faite au sujet de la définition, et il importe de bien la comprendre pour saisir le danger réel des virus informatiques. Il s'agit de l'observation que *la propriété d'infection est inhérente aux virus.* En clair, cela signifie qu'un virus n'est pas un programme qui exploite un défaut (erreur ou omission) d'un système d'exploitation. C'est au contraire un programme normal, appartenant à un utilisateur normal, et n'utilisant que des opérations normales que d'autres programmes utilisent également. Par exemple, il n'est pas nécessaire qu'un virus "pirate" la table des mots de passe d'un système afin d'y pénétrer. Il lui suffit, comme nous le verrons, d'attendre qu'un utilisateur l'exécute pour hériter de ses droits d'accès et avoir ainsi l'occasion d'infecter les programmes sur lesquels il a un droit de modification. Nous pouvons affirmer qu'un système peut être infecté par un virus dès qu'un partage d'information y est autorisé [COH-88].

1.3. Historique des virus et virus biologiques

Nous poursuivons ce bref exposé de la définition d'un virus en brossant un rapide historique de leur existence [SCH-88]. Nous le terminerons en examinant le bien-fondé de l'emploi de la terminologie médicale dans ce domaine particulier de l'informatique.

Historiquement, le principe des virus apparaît dès 1949 dans l'ouvrage de J. Von Neumann intitulé *Theory and organization of complicated automata* [NEU-49]. Vingt-cinq années plus tard, se basant sur les recherches effectuées lors de la conception de systèmes multitâches comme UNIX, apparaît le jeu "Core War". Ici, deux joueurs écrivent des programmes auto-reproducteurs dont le but est de dévorer les programmes de l'adversaire. Il ne s'agissait

pas encore de virus selon la définition que nous en avons donnée, car aucune infection n'était alors réalisée; il n'en reste pas moins que l'entrée de ce "jeu" dans le domaine public a certainement inspiré les premiers créateurs de virus, en conjugaison avec des travaux sur les "distributed computations" (voir note 4, page 32).

En examinant les "virus", ces programmes "infectieux", on peut remarquer certaines analogies de "comportement" avec les virus biologiques. La nécessité de "parasiter" pour survivre, la capacité de se reproduire, la contagion par contact avec un organisme infecté (biologique ou informatique), tous ces indices nous confortent dans notre impression première : l'homme a créé l'équivalent informatique des virus biologiques. Or, cette comparaison ne résiste pas à une analyse plus poussée.

Un virus biologique est une séquence d'acide nucléique (ARN ou ADN); cet acide nucléique contient notamment le code de la duplication du virus. Un virus ne contient donc pas de machinerie cellulaire qui lui permettrait de synthétiser des protéines et de fabriquer de l'énergie. En conséquence, il est incapable de croissance et de division, ce qui en fait un parasite obligé [BUR-86]. Il porte préjudice à la cellule infectée, pouvant même aller jusqu'à sa mort lorsque celle-ci éclate pour laisser sortir les virus matures.

A l'opposé, un virus informatique est un programme à part entière, qui pourrait s'exécuter indépendamment de tout autre programme. Il ne "parasite" d'autres programmes que parce qu'il est conçu pour cela, et pas pour survivre. De plus, il n'a pas besoin d'obliger le programme infecté à faire quelque chose pour pouvoir se reproduire, il contient lui-même toutes les instructions permettant de le faire.

Un virus informatique n'a donc pas les caractéristiques d'un virus biologique; tout au plus utilise-t-il le programme porteur pour se dissimuler et pour bénéficier de ses droits d'accès. Il ne lui porte préjudice en aucune manière, et ne l'oblige pas à lui sacrifier ses propres fonctions. Nous continuerons cependant à utiliser cette terminologie, car c'est elle qui est utilisée dans la littérature. Considérons simplement chacun de ces termes comme l'abréviation d'un terme "composé"; ainsi, nous dirons "virus" pour "virus informatique", etc.

1.4. Conclusion

Après avoir défini les virus et commenté cette définition, nous avons montré que la comparaison entre les virus informatiques et les virus biologiques qui fait "saliver" quelques journalistes "à sensation" (1) n'est pas plus pertinente que d'affirmer qu'un automate joue aux échecs comme Karpov.

2. Déroulement d'une attaque virale

Dans cette section, nous décrivons les trois phases de l'attaque d'un virus, qui constituent en quelque sorte son "cycle de vie". Ce sont la *contamination*, la *contagion* et le *déclenchement* (2).

Le terme *victime* sera utilisé dans la suite pour désigner un utilisateur dont au moins un fichier est infecté par un virus dont il n'est pas l'auteur. Le terme *sain* se rapporte à un utilisateur qui n'a pas écrit de virus et qui n'a été victime d'aucun virus.

2.1. La contamination

Elle a lieu au moment où un utilisateur sain devient la victime d'un virus. Il s'agit par exemple d'un utilisateur quelconque se procurant un programme infecté, ou d'un utilisateur de "mainframe" ayant accordé des privilèges de modification sur ses fichiers aux membres de son groupe ou à ses correspondants à travers le réseau. Retenons simplement que l'ensemble des fichiers sur lesquels la victime a un droit d'exécution contient au moins un fichier infecté.

2.2. La contagion

La victime, ignorant la présence du virus, exécute un programme infecté. Le virus en profite alors pour se recopier dans d'autres fichiers. Si la victime n'agit pas pour le stopper, le processus de contagion fait boule

---

1) [GRU-88] n'est qu'un exemple parmi tant d'autres ...

2) Ces trois phases ont été dégagées à la suite d'une discussion avec Mr. FERREIRA. A notre connaissance, ce découpage est inédit dans la littérature.

de neige : plus la victime exécute de programmes infectés, plus le virus dispose d'opportunités pour infecter d'autres programmes.

Remarquons ici que les virus ont la propriété de persister indéfiniment. Toute copie de sauvegarde d'un fichier contaminé est en effet elle-même infectée, ce qui peut induire une recontamination d'un système désinfecté lorsqu'une telle copie de sauvegarde est réutilisée dans ce système. La contagion peut donc avoir lieu jusque dans les supports externes qui contiennent ces copies de sauvegarde.

2.3. Le déclenchement

Nous n'avons pas encore mentionné cette caractéristique qu'ont certains virus et qui les fait tant redouter. En effet, un virus peut être porteur de n'importe quelle information; il peut donc être utilisé pour causer des effets arbitraires. Nous appellerons "déclenchement" l'exécution de cette action arbitraire.

Le déclenchement peut avoir lieu à tout moment du processus de contagion; ce moment dépend bien sûr de la stratégie adoptée par le concepteur du virus.

Remarquons à nouveau ici qu'un virus n'est pas nécessairement "nuisible", même si la plupart des virus ont été jusqu'ici écrits dans ce but (3). Un virus peut même ne jamais se déclencher, soit parce que la condition qui pilote ce déclenchement ne sera jamais atteinte, soit parce qu'aucune action n'a été prévue par son auteur (voir par exemple le virus décrit à la figure 3.1. et qui ne comporte pas de déclenchement).

En résumé, une attaque virale débute par la contamination de la victime. Une fois celle-ci en contact avec le virus, la contagion peut se produire. Un éventuel déclenchement complète cette attaque, dont les suites ne sont pas nécessairement à redouter.

3. Analyse du processus d'infection

Nous avons déjà fait ample usage de cette notion d'infection dont nous

---

3) Notons à ce propos la connotation "agressive" du terme *attaque* utilisé pour désigner ce phénomène de pénétration. Mais comme il est largement utilisé dans la littérature ([COH-87, COH-88, POZ-86] par exemple) il nous a semblé préférable de le conserver.

avons dit qu'elle suffit à définir un virus. Nous allons tout d'abord la préciser, puis l'étudier plus en détail en prenant en compte l'environnement dans lequel elle a lieu. Enfin, nous examinerons le problème des *virus évolutifs* ou *mutants*.

3.1. Définition et déroulement d'une infection

Nous avons dit plus haut que l'infection avait pour but de recopier le code du programme "infectant" dans celui du programme "infecté", la copie du code pouvant subir des modifications en cours de processus. Cette définition laisse en suspens le "comment" de l'infection.

Attachons-nous en premier lieu au déroulement de l'infection. Celui-ci est schématisé à la figure 3.2.

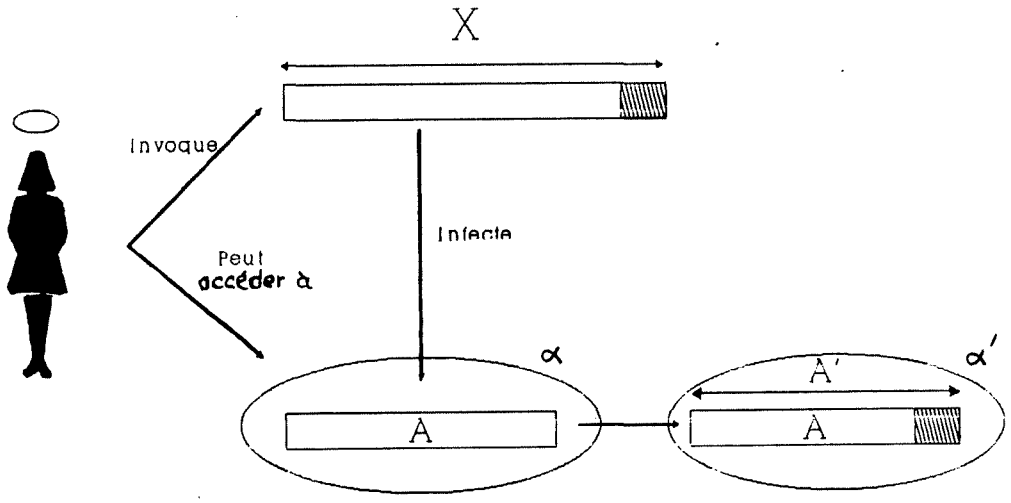

Figure 3.2. : Le processus d'infection

La future victime possède un droit de modification sur un ensemble α de fichiers, soit parce que les propriétaires de ces fichiers le lui ont accordé, soit parce qu'elle le possède d'office ("Super user" de UNIX, par exemple), soit parce que les fichiers lui appartiennent, ou pour toute autre raison. Elle possède également un droit d'exécution sur un fichier X, dont elle ignore qu'une partie du code est celui d'un virus. Elle est donc déjà contaminée, selon la définition donnée plus haut.

Lorsque le programme X est exécuté, le code du virus est également exécuté, en plus du service "normal" fourni par une copie saine de X. Le virus, exécuté par la victime, hérite de ses droits; en particulier, il a le

droit de modifier tout fichier de α. L'exécution du virus a pour effet d'insérer son code dans au moins un des fichiers de α, par exemple A. Nous appellerons A' le fichier A ainsi modifié, et α' l'ensemble des fichiers défini par : $\alpha' = (\alpha \setminus \{A\}) \cup \{A'\}$.

Nous pouvons alors dire que la contagion aura lieu lorsque la victime commencera à exécuter d'autres programmes infectés, comme A'. Ce processus de contagion ne se terminera que lorsque tous les fichiers de α' seront infectés. Un éventuel déclenchement peut avoir lieu à tout moment de l'exécution du virus.

3.2. Commentaires

Voyons d'abord comment ce processus s'est enclenché.

Le créateur du virus, comme expliqué à la figure 3.3., inclut son virus dans un programme Y, que nous appellerons alors Y'. L'exécution du programme Y' provoquera une infection tout à fait classique, expliquée ci-dessus. Le processus doit donc être enclenché "manuellement".

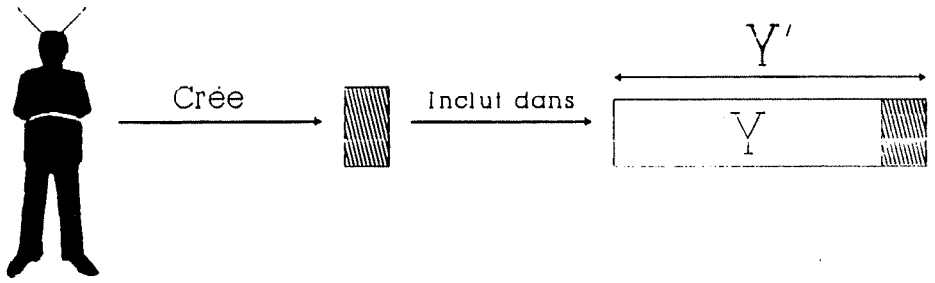

Figure 3.3. : Le lancement du processus

Nous avons également mentionné le choix d'un "certain programme" A comme cible de l'infection. Remarquons que la stratégie adoptée par le concepteur du virus pour réaliser ce choix influera beaucoup sur la capacité du virus à passer inaperçu. Il est par exemple possible à un virus d'infecter en une seule fois tous les fichiers auxquels il a accès au moment de son exécution. Mais dans ce cas, les performances du programme infecté en cours d'exécution seraient tellement dégradées qu'il deviendrait pratiquement impossible à un utilisateur de ne pas s'apercevoir que quelque chose d'anormal est en train de se passer !

Pour clôturer ces quelques remarques, notons qu'il est possible que le

processus de contagion se termine avant l'infection du dernier fichier de α'. En effet, un utilisateur s'apercevant qu'il est contaminé pourrait prendre des mesures de désinfection, par exemple en lançant un programme voyageur (4) dont le but est de trouver et de détruire le virus repéré. Notons encore que cette désinfection, pour être efficace, doit avoir lieu sur toute la fermeture transitive du partage d'information entre utilisateurs. En effet, le domaine d'infection d'un virus est constitué de toute la fermeture transitive du partage d'information entre utilisateurs, c'est-à-dire l'ensemble de toutes les cibles qu'une information donnée est susceptible d'atteindre en empruntant n'importe quel chemin qui lui est autorisé. Plus formellement, nous définirons la fermeture transitive du partage de l'information d'un objet (5) f comme suit :

$$FT(f) = C(f) \cup \left(\bigcup_{f_1 \in C(f)} FT(f_1) \right)$$

où $C(f) = \{f\} \cup \{$objets f_1 tels que f peut partager de l'information avec $f_1\}$.

Cette définition découle de deux propriétés essentielles de l'information : la *transitivité* et la *généralité de l'interprétation*, associées à la *possibilité de partager de l'information*. La propriété de transitivité est la capacité qu'a l'information de pouvoir passer d'un utilisateur A vers un utilisateur C si elle peut passer de A vers un utilisateur B et de B vers C. La généralité d'interprétation signifie qu'il n'y a pas de distinction fondamentale entre l'information utilisée comme donnée et celle pouvant être utilisée comme programme. L'information n'a en effet de sens que dans le contexte d'une interprétation particulière [COH-87].

---

4) Un "WORM"; il s'agit d'un programme qui a la propriété de s'exécuter sur plusieurs machines à la fois, en gérant lui-même sa "reproduction" de site en site pour décharger les machines fort utilisées en reportant le travail sur des hôtes moins "occupés" [SHO-82]. Notons qu'à cause d'une erreur de programmation, un programme de ce type a entièrement saturé le réseau ARPA en novembre dernier [ACM-89].

5) Dans la suite, le mot "objet" fera référence à une entité cohérente d'information (comme un fichier ou un programme, par exemple).

Dans le contexte d'une infection, le "partage d'information " doit être pris dans le sens suivant :

Un objet f peut partager de l'information avec un objet f_1 si d'une part il existe une interprétation I dans laquelle l'objet f peut agir de manière virale, et si d'autre part cet objet f a la possibilité de modifier l'objet f_1 dans cette interprétation I.

La notion de fermeture transitive du partage d'information nous fournit un outil d'abstraction permettant d'étudier la sécurité des systèmes indépendamment d'une configuration particulière. Dans la pratique, il est toutefois utile d'examiner également la configuration particulière à protéger; c'est pourquoi nous présentons ci-après quelques exemples d'environnements largement utilisés.

3.3. Types d'environnements [FAK-88]

Nous exposons ici les risques de contamination et de contagion dans différents systèmes. Après avoir énoncé les risques pesant sur les environnements P.C., mini-ordinateur et ordinateur central, nous mentionnerons les problèmes des réseaux.

3.3.1. Environnement clos

Le premier type d'environnement envisagé est le P.C. "stand-alone", c'est-à-dire non relié à un réseau. Le risque de contamination le plus important sur ce type de configuration est de loin la possibilité de charger n'importe quel programme dans le système au moyen d'une disquette. Cela rend les P.C. très vulnérables, à cause du vaste brassage de programmes dans le monde entier, par l'intermédiaire notamment des "clubs informatiques". De plus, l'information nécessaire à la construction d'un virus est très facile à obtenir et il est aisé de trouver un endroit tranquille, propice aux expérimentations de toutes sortes.

Une fois le virus entré dans le système, il a théoriquement accès à tous les fichiers de chacun des disques de ce système. Rien n'empêche donc la contagion des fichiers d'un environnement P.C. standard une fois celui-ci contaminé.

Nous constatons donc que cet environnement est extrêmement vulnérable à une attaque virale. La configuration composée d'un mini-ordinateur partagé par plusieurs utilisateurs offre-t-elle une proie moins facile aux virus ? Pour le vérifier, nous allons opérer une classification des types d'utilisations de ce genre de configuration; chacune des classes sera alors analysée séparément.

Deux classes peuvent être distinguées : la première comprend les ordinateurs ne fournissant des services qu'à des utilisateurs non programmeurs, avec très peu de changements dans l'ensemble des programmes; la seconde comprend les ordinateurs dans un contexte de développement, où la programmation est une activité plus fréquente.

La première classe s'apparente aux ordinateurs centraux : les contrôles sont rigides, peu d'utilisateurs ont le droit d'ajouter de nouveaux programmes, etc. Les risques dans cette classe sont similaires à ceux qui menacent les plus gros ordinateurs.

La seconde classe, par contre, s'apparente davantage aux P.C. : ces systèmes sont très peu cloisonnés car le partage d'information y est maximal. Ce décloisonnement est en général voulu par les utilisateurs, mais il a pour effet de restreindre fortement la possibilité qu'une partie du système puisse demeurer saine alors qu'une autre partie de ce système est déjà contaminée. Cette opportunité est de plus limitée par l'existence d'un *super utilisateur* ou *gestionnaire du système* (concept UNIX du *super user*), qui possède tous les droits sur tous les fichiers du système. Ce super utilisateur étant en général un utilisateur effectuant les mêmes travaux que les autres, le danger est qu'un programme infecté soit exécuté par lui. Dans ce cas le virus hériterait de tous les privilèges nécessaires à l'infection des derniers "bastions" sains. Ce concept nous indique qu'un cloisonnement dans une telle machine partagée n'interdira pas au virus d'infecter tous les recoins de cette machine; la contagion n'en sera que ralentie dans la majorité des cas.

De même que dans un environnement P.C., la contamination peut avoir lieu par importation d'un programme infecté, mais également de l'intérieur, un utilisateur malveillant pouvant lui-même créer un virus. Ce risque dépend beaucoup du fonctionnement de l'organisation visée mais ne peut être négligé dans aucun des cas. Le risque d'importation d'un virus, quant à lui, paraît plus limité que dans une configuration de type P.C. Le brassage de programmes semble en effet plus restreint sur ce type de machine; ce fait est probablement dû à la différence de profil des utilisateurs (il y a peu de

clubs informatiques d'utilisateurs de mini-ordinateurs, par exemple), ainsi qu'au caractère peu pratique des supports employés (bande magnétique, etc). Un danger bien plus grand apparaîtra avec la connexion à un réseau.

Le dernier type d'environnement clos que nous envisagerons est celui constitué d'un gros ordinateur central auquel sont connectés jusqu'à plusieurs centaines de terminaux. Malgré la présence d'une énorme quantité d'information sur ces machines, il semble qu'elles soient mieux protégées que les autres, grâce à un cloisonnement plus efficace associé à des mesures de sécurité spéciales. Ainsi, le développement est strictement séparé de l'utilisation journalière. Des procédures plus strictes sont utilisées, notamment dans les politiques d'ajout de nouveaux programmes, qui seraient d'ailleurs aussi utiles pour améliorer également la sécurité de systèmes plus petits.

Le danger principal se présente ici sous deux aspects. Le premier est celui de la transmission d'un virus de l'environnement de développement à l'environnement de production, là où un maximum de dégâts peut être causé à l'organisation visée. Le développement est en effet très vulnérable pour les mêmes raisons que les mini-ordinateurs. Le second aspect du danger réside ici aussi dans l'existence d'un et parfois même de plusieurs super utilisateurs.

Un cloisonnement n'est donc pas beaucoup plus efficace à empêcher la contagion d'un gros ordinateur central qu'il ne l'était pour empêcher celle d'un mini-ordinateur.

Remarquons de plus que le danger est ici surtout interne. En effet, les possibilités d'infection provenant de l'extérieur diminuent avec l'efficacité des mesures de contrôle, alors que le risque d'infection causée par un utilisateur légitime du système augmente avec le nombre d'utilisateurs. Il n'en reste pas moins qu'il serait dangereux de négliger le risque d'infection externe, car même si la probabilité d'une infection a diminué, elle ne deviendra nulle que lorsque tout partage d'information aura été rendu impossible.

Nous avons donc vu que, dès qu'un partage d'information est autorisé dans un système, le risque d'une contamination par un virus existe. Cette contamination est rendue possible par la transitivité et par la généralité d'interprétation de l'information . La contagion, quant à elle, ne pourra être inconditionnellement limitée que lorsque le concept de super utilisateur aura disparu du système à cloisonner.

3.3.2. Environnement ouvert

L'étude approfondie de la propagation des virus dans les réseaux sort du cadre de ce mémoire. Le lecteur intéressé pourra se référer aux travaux de F. COHEN (6). Nous limiterons notre propos à ce qui suit.

Les réseaux offrent par définition un moyen efficace de se partager de l'information; par conséquent, ils peuvent être utilisés par un virus qui s'en servira comme véhicule pour réaliser plus rapidement l'infection d'un nombre plus important de machines (voir figure 3.4.). Ce virus doit évidemment avoir été conçu de telle sorte qu'il sache se servir des procédures d'accès.

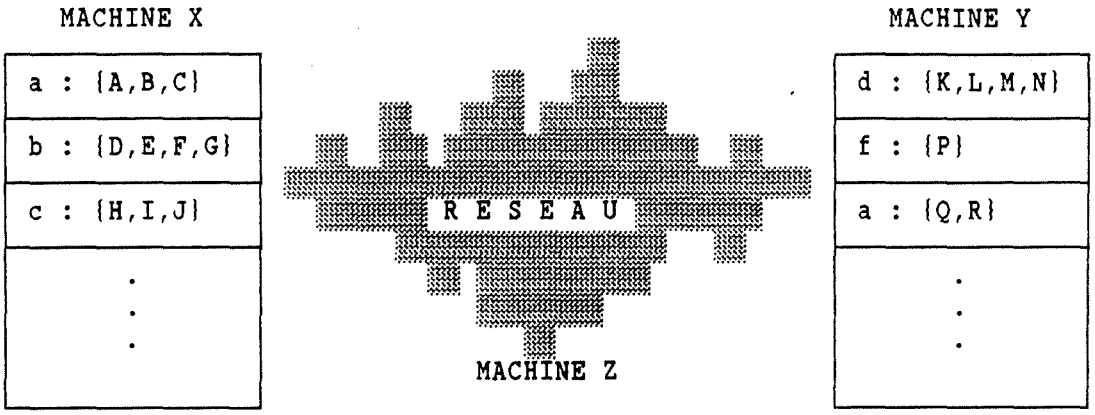

Figure 3.4. : Contamination au travers d'un réseau

Exemple 1 : L'utilisateur a, connecté sur la machine X, fait exécuter le programme A sur la machine Y. Si A est infecté, il peut contaminer des objets sur la machine Y, par exemple Q.

Exemple 2 : L'utilisateur f, connecté sur Y, a le droit de modifier des objets appartenant à b sur la machine X. Si P, contaminé, est exécuté par f sur Y, il a le droit de modifier les objets de b sur X et donc d'infecter par exemple G.

Remarquons encore qu'un réseau n'est qu'un moyen supplémentaire, parfois très efficace, de se partager de l'information. La présence du

---

6) Voir par exemple [COH-85], ainsi que les articles qu'il a publiés à ce sujet dans le volume 6 de la revue *Computers & Security*, p. 118-143 et p. 332-338.

réseau n'augmente en effet la capacité théorique d'infection du virus que dans la mesure où aucun autre moyen n'existe pour lui d'atteindre toutes les machines visées. Par exemple, connecter un P.C. à un réseau n'augmente le nombre de programmes susceptibles de l'infecter que s'il l'autorise à partager de l'information avec des ordinateurs alors qu'il était dans l'impossibilité de le faire avec certains d'entre eux avant l'introduction de ce réseau. En d'autres termes, nous pouvons affirmer que l'introduction d'un réseau n'étend pas forcément la fermeture transitive du partage de l'information d'un virus; s'il l'étend, c'est qu'une décision a été prise d'autoriser un partage d'information à un endroit où c'était auparavant impossible.

Il est néanmoins correct d'affirmer que les dégâts sont plus importants en présence d'un réseau, en général tout au moins. Cette observation s'explique par la vitesse de transfert offerte au virus, qui l'autorise dans certains cas à infecter un grand nombre de machines quasi-instantanément. Nous pouvons donc dire que le réseau, s'il n'étend pas la fermeture transitive du partage d'information, fournit au virus une opportunité supplémentaire d'en infecter un maximum d'objets.

Ce bref exposé des risques dans les environnements P.C., mini-ordinateur, ordinateur central et réseau, a permis de montrer que le concept de fermeture transitive était un outil d'abstraction intéressant pour étudier les virus indépendamment d'une configuration particulière. Nous pouvons alors conclure en disant que le seul fait de partager de l'information entre systèmes nous interdit de prétendre, dans ces systèmes, à la protection inconditionnelle contre les virus.

Nous allons à présent examiner une caractéristique des virus, qui leur permet d'évoluer lors de l'infection.

3.4. Les virus évolutifs [COH-87]

Notre but est ici de démontrer qu'il est possible de construire un virus dont le contenu du code n'est pas constant de génération en génération, mais qui reste cependant "lui-même" (c'est-à-dire que les deux versions sont équivalentes, ou encore qu'elles ont le même effet).

La manière la plus évidente de démontrer que quelque chose existe est de le construire. C'est ce qui est réalisé à la figure 3.5. Le code "recopié" du virus est rendu différent de son code "source" par l'insertion aléatoire d'instructions aléatoires non nécessaires entre des instructions

nécessaires.

La possibilité qu'un virus puisse évoluer lors du processus d'infection est donc démontrée. Cette évolutibilité rend ces virus beaucoup plus difficile à détecter, car une détection automatique revient à établir l'équivalence de deux versions d'un programme. Ce problème ayant été démontré indécidable [LER-87, COH-87] nous nous trouvons ici devant un obstacle majeur à la désinfection automatique "polyvalente" (lorsque le code du virus est inconnu a priori). Voir à ce propos la section consacrée aux moyens de désinfection.

```
REPETER
      fich <- choisir_un_programme_au_hasard;
JUSQU'A (pas_encore_infecte (fich));
ajout_virus_avec_insertions (fich);
aller_a (debut_programme_infecté);


PROCEDURE ajout_virus_avec_insertions (f);
      TANT QUE (fin_virus = FAUX) FAIRE
            SI (bit_au_hasard = 1)
            ALORS  ajout_instruction_au_hasard (f);
            ajout_ligne_suivante_du_virus (f);


PROCEDURE ajout_instruction_au_hasard (f);
      ajout_mot (variable_au_hasard, f);
      ajout_mot ("<-", f);
      ajout_mot (variable_au_hasard, f);
      TANT QUE (bit_au_hasard = 1) FAIRE
            ajout_mot (operateur_au_hasard, f);
            ajout_mot (variable_au_hasard, f);
      ajout_mot (";", f);
```

Figure 3.5. : Algorithme d'un virus évolutif

3.5. Conclusion

Nous nous sommes attachés à préciser la propriété d'infection des virus. Après avoir expliqué ce processus d'infection, nous l'avons examiné dans différents environnements. Nous en avons conclu qu'indépendamment d'une configuration particulière, les risques de contamination et de contagion existent dès qu'un partage d'information est autorisé. Ces risques varient en fonction des moyens utilisés par le virus lors de l'infection.

En sus des risques cités, un problème supplémentaire apparait au responsable de la sécurité : la détection des virus est rendue plus ardue par la capacité de ceux-ci d'évoluer de génération en génération.

4. Le déclenchement

La notion de déclenchement énoncée plus haut est ici brièvement développée.

Nous avons déjà vu qu'il était possible d'inclure dans un virus n'importe quelle information, qui sera interprétée comme une action particulière à exécuter pourvu qu'une certaine condition soit vérifiée. Cette action est appelée "déclenchement"; de même, la condition sera appelée "condition de déclenchement".

Pour illustrer ce propos, nous pouvons modifier le virus de la figure 3.1., de telle sorte qu'il exécute maintenant une action prédéfinie après l'infection chaque fois qu'une condition prédéterminée est vérifiée. Cet algorithme est présenté à la figure 3.6.

```
REPETER
      fich <- choisir_un_programme_au_hasard;
JUSQU'A (pas_encore_infecte (fich));
ajout_virus_au_fichier (fich);
Si (condition_de_declenchement = VRAI)
ALORS
      se_declencher;
aller_a (debut_programme_infecté);
```

Figure 3.6. : algorithme d'un virus avec déclenchement

Cette propriété donne la possibilité au créateur de faire exécuter une action qu'il a lui-même définie partout où le virus accédera, sans nécessairement y avoir accès personnellement (cette action sera programmée dans la procédure se_declencher). Or, ainsi que nous l'avons vu précédemment, un virus a accès à toute sa fermeture transitive du partage d'information. Il en résulte que les virus constituent un danger très grave pour les systèmes informatiques, car la seule manière de leur interdire inconditionnellement tout accès à un système est d'isoler complètement celui-ci en interdisant tout partage d'information. Cette solution étant inacceptable dans l'immense majorité des cas, nous ne disposons pour nous protéger que des méthodes de prévention et de désinfection dont l'efficacité absolue ne pourra jamais être garantie.

5. Conclusion

Les virus informatiques sont définis par leur capacité d'infecter d'autres programmes. L'infection a pour domaine la fermeture transitive du partage d'information du virus envisagé, ce qui nous permet de les étudier indépendamment d'une configuration particulière.

Le danger réel de contamination virale est plus grand lorsque des moyens rapides sont utilisés pour se partager l'information . Les virus, capables d'utiliser ces moyens rapides pour se répandre puis de se déclencher une fois dans la place, font ainsi peser une menace non négligeable sur les systèmes informatiques.

Pour protéger un système contre cette menace, la seule solution inconditionnellement sûre est d'empêcher tout partage d'information entre ce système et le monde extérieur. Il faut donc envisager la possibilité d'une contamination de tout système où l'on désire partager de l'information avec l'extérieur, il faudra par conséquent prendre des mesures de prévention et de désinfection. Ces mesures devraient être assorties de techniques destinées à empêcher la contagion au sein d'un système contaminé, par exemple un cloisonnement efficace sans super utilisateur.

DEUXIEME PARTIE

LES PROTECTIONS

La malveillance informatique n'est pas un problème qui peut être considéré comme une fatalité technique dont il faut prendre son parti. Il nous semble en effet qu'un acte commis par un être humain ne peut pas être contré efficacement par les seuls moyens techniques. Une approche plus globale, incluant les aspects juridique, organisationnel et déontologique, paraît plus à même de prendre en compte les divers aspects de ce qui peut être appelé la *criminalité informatique*.

La seconde partie de ce travail est consacrée aux moyens que l'on peut mettre en oeuvre pour "guérir" mais également pour "prévenir" les problèmes introduits par la malveillance informatique. Par l'application de mesures légales, organisationnelles et déontologiques, on vise à protéger l'ensemble des ressources informatiques, à conserver l'intégrité de l'information contenue dans le système, à préserver le caractère confidentiel de cette information et à assurer le fonctionnement ininterrompu des systèmes informatiques.

Nous avons choisi d'envisager plus particulièrement leur application dans le domaine de la protection des logiciels contre la malveillance. Mais insistons sur le fait que l'ampleur des solutions mises en oeuvre devra être directement proportionnelle au niveau des risques encourus par le système que l'on veut protéger d'une part, et d'autre part au niveau de sécurité que l'on veut assurer.

CHAPITRE I

Solutions légales, Organisationnelles et Déontologiques

CHAPITRE I : PROTECTIONS JURIDIQUES, ORGANISATIONNELLES ET DEONTOLOGIQUES

Ce chapitre a pour but de lever un coin du voile sur les solutions à caractère "non technique" de protection des logiciels contre la malveillance. Des solutions juridiques seront d'abord esquissées, suivies de solutions socio-organisationnelles. L'amorce d'une approche déontologique du problème sera ouverte en fin du chapitre.

Nous nous inspirons fortement ici d'un travail réalisé par d'autres étudiants [HEU-89].

1. Solutions juridiques

La littérature montre que les recherches en la matière s'orientent plutôt vers le droit pénal (nouveau ou adapté) au détriment du droit civil.

1.1. Le recours au droit pénal

En matière de droit pénal, les législations existantes s'avèrent souvent insuffisantes pour lutter efficacement contre la plupart des délits informatiques. Dès lors, certains prônent la création d'une législation spécifique à l'informatique tandis que d'autres suggèrent une adaptation du droit actuel. Ces deux courants peuvent être illustrés respectivement par "l'exemple" belge et "l'exemple" canadien.

Catherine ERKELENS (VUB) montre que, dans de nombreux cas, le droit belge actuel est inapplicable. Par exemple, la manipulation de données ne peut être assimilées à un *faux en écriture* que si ces données peuvent être considérées comme un écrit au sens juridique du terme; le détournement et l'obtention frauduleuse d'informations sont, quant à eux, non punissables. Ceci mène donc à la conclusion qu'il est nécessaire d'avoir recours à une législation spécifique : *un droit de l'informatique.*

Le sous-comité canadien sur les infractions relatives aux ordinateurs

[CJJ-83] qui s'est occupé de cette question estime quant à lui que la législation actuelle suffit pour punir les délits informatiques dans le cas où l'ordinateur est l'instrument du délit (fraude, ...) ou lorsqu'il en est l'objet (sabotage, ...).

Il existe cependant des cas non prévus par la loi tels que l'entrée illicite dans un système ou la copie non autorisée de données ou de programmes. Ces délits sont spécifiques à la technologie informatique; ils ne pourraient pas exister sans l'ordinateur alors que les premiers types de délits existaient avant son l'apparition. Pour remédier à ce problème, le sous-comité propose une modification du droit pénal visant à protéger l'inviolabilité des ordinateurs. Mais la prudence s'impose dans l'élaboration de ces mesures; l'informatique étant une discipline en continuelle évolution, il conviendra d'axer les définitions nécessaires à la description des infractions sur les opérations et non sur les techniques en cause.

Un autre problème spécifique à l'informatique se pose encore; c'est celui de la preuve. Peut-on, par exemple, considérer un listing comme un écrit au même titre qu'un document original ? Une proposition de loi a été faite dans ce sens. Mais quelle protection peut attendre un utilisateur victime d'un programme lui ayant porté préjudice avant d'effacer toute trace de son passage (par exemple, un virus programmé pour s'autodétruire après un certain temps) ? Le problème de la preuve se montre ici dans toute son ampleur.

1.2. Le recours au droit civil

Parallèlement au droit pénal, beaucoup d'auteurs s'accordent à dire qu'un recours au droit civil devrait être prévu afin de permettre à la victime d'obtenir un dédommagement. La punition du coupable n'est bien sûr pas suffisante de ce point de vue.

1.3. Le problème de la preuve

Parallèlement à la législation, il faut donc mettre en place des moyens d'investigation permettant de réunir les preuves du délit. A ce niveau, nous rencontrons deux types de problèmes. Le premier d'entre eux est un problème technique, qui a trait à l'expertise; la technologie exige que l'on dispose d'experts pour assister la magistrature. Les autres problèmes sont de type

légal. Les enquêteurs doivent disposer de pouvoirs spéciaux d'accès (saisie, perquisition, ...) pouvant aller jusqu'à une coopération forcée du propriétaire du système. De plus, la législation et les moyens d'investigation de la poursuite judiciaire doivent être harmonisés au niveau international. En effet, suite à l'abolition des frontières et au développement des réseaux internationaux, on assiste à une internationalisation de la criminalité en général et de la criminalité informatique en particulier. Une telle harmonisation aurait pour but d'éviter que certains pays ne deviennent des pôles de criminalité. Elle permettrait aussi, par des traités de coopération entre états, d'étendre le champ d'investigation au niveau international.

1.4. L'absence de jurisprudence

Un dernier point important est l'absence de jurisprudence en matière de délits informatiques. Cette carence est due d'une part à l'originalité de ces problèmes, et d'autre part au fait que dans la plupart des cas il n'y a pas de plainte. En effet, si la victime est une entreprise, elle juge souvent dangereux pour sa réputation de dévoiler un tel secret. Le règlement du problème se fera alors de manière interne, si c'est toutefois possible !

1.5. Conclusion

La solution juridique combine des caractéristiques préventives (car elle met en évidence les conséquences de l'acte) et répressives (car elle offre les moyens d'actions légaux pour poursuivre les auteurs).

Deux tendances prédominent actuellement pour apporter une réponse juridique aux nouveaux problèmes posés par l'informatique : l'adaptation du droit existant ou la création d'une législation spécifique.

2. Solutions socio-organisationnelles

> *De loin les plus importantes, les mesures organisationnelles représentent un ensemble de fonctions de gestion des ressources matérielles et humaines.* [VIT-89]

Ce type de solutions consiste en l'application partielle ou totale des mesures suivantes :

1. mise en place d'une structure organisationnelle basée sur l'éthique et la morale : *The best instructors in ethics and morality are those who set good examples. Example is a sine qua non in ethical learning* [MOW-86];
2. formation adéquate du personnel : *Rules, norms and guidelines are necessary to define what one ought and ought not do* [MOW-86];
3. dialogue avec le personnel et actions de motivation;
4. analyse des risques par construction de "scénarios catastrophes" et analyse coûts/risques;
5. instauration de méthodologies de conception permettant les contrôles;
6. savoir qui fait quoi, qui a accès à quoi;
7. distribution des responsabilités afin de s'assurer qu'une personne n'ait pas une liberté d'action trop étendue lorsque celle-ci n'est pas nécessaire.

Ces mesures sont pour la plupart préventives mais certaines d'entre elles peuvent être perçues comme étant répressives pour le personnel (comme par exemple les mesures 3 et 7 et dans une moindre part, la mesure 5).

3. Solutions déontologiques

Il s'agit ici de mettre en place un ensemble de règles de conduite relatives au comportement des informaticiens, aussi bien vis-à-vis des employeurs que des clients, de l'organisation ou de la profession.

Une récente étude en criminologie menée au Québec montre que les délits informatiques ne sont pas d'abord le fait d'individus souffrant de problèmes socio-économiques, psychologiques ou autres, mais d'individus témoignant d'un manque d'éthique professionnelle. Cette étude montre de plus que de façon générale ces individus nient le caractère illégal de leur conduite, la situant au niveau supérieur de la connaissance, de l'expérience ou de la recherche.

Pour augmenter l'efficacité des contrôles technologiques, Danielle POUILLOT [VIT-89] préconise de renforcer la notion d'éthique professionnelle en développant dans chaque organisation un code de conduite et d'y associer des sanctions en cas de non respect.

Il est certain que ce type de solution n'offre pas une garantie absolue quant à l'intégrité des adhérents au code de conduite, notamment parce que les sanctions ne peuvent être, souvent, que des condamnations de principe.

L'aspect essentiellement préventif des codes de conduites s'insère dans le cadre plus général d'une régulation du comportement des informaticiens. Cette solution concerne tous les types de fraude qui pourraient être commis par les informaticiens (ainsi que leur responsabilité vis-à-vis des tiers) mais ne touche en aucun cas les personnes extérieures à la profession.

4. Conclusion

L'approche juridique de protection des logiciels, plutôt dissuasive, met l'accent sur la punition du coupable (droit pénal) et sur l'indemnisation de la victime (droit civil); mais les problèmes de la preuve et de la nouveauté de la technique informatique rendent souvent ces solutions légales difficiles à appliquer. L'approche organisationnelle a pour but de lutter contre la fraude des "cols blancs", notamment en instituant un certain nombre de contrôles; ceux-ci peuvent être mal ressentis par les personnes auxquel*les* ils s'appliquent, et systématiquement contournés. L'approche déontologique vise à conscientiser les informaticiens de leurs responsabilités; mais certaines personnes peuvent refuser de se laisser conscientiser.

Ces trois approches complémentaires sont donc insuffisantes dans la majorité des cas à protéger efficacement les logiciels contre la malveillance; tout au plus parviendra-t-on à la réduire. La nature humaine restant ce qu'elle est, il faudra dès lors compléter ces solutions "non techniques" par des mesures techniques dont l'objectif sera de prévenir la malveillance résiduelle.

CHAPITRE II

Protections techniques

CHAPITRE II : PROTECTIONS TECHNIQUES

Les moyens techniques peuvent être utilisés à quatre niveaux dans un but de sécurisation des logiciels.

Le premier de ces niveaux consiste, pour un utilisateur, à protéger son droit d'exclusivité sur le code dont il est le possesseur légitime; ce code sera qualifié de confidentiel. Le deuxième niveau apparaît "à la source", lorsque le distributeur légitime du logiciel met en oeuvre certains moyens de protéger son droit. Le troisième niveau comprend la protection de l'intégrité du logiciel, mise en oeuvre par le propriétaire légitime de celui-ci. Le dernier niveau que nous envisagerons consiste à protéger l'accès au logiciel. En guise d'illustration, une brève présentation de la carte à microprocesseur sera donnée en fin de chapitre, ainsi que des potentialités qu'elle offre (notamment pour le contrôle d'accès).

Mais comme la plupart de ces solutions font appel à la cryptographie, nous en présentons quelques notions de base en guise d'introduction.

1. Quelques notions de base en cryptographie

1.1. Définitions

La cryptographie consiste à faire subir des transformations syntaxiques aux messages au moyen d'un jeu de clés, de manière à les rendre incompréhensibles tout en étant capable à tout moment de les restituer en clair par le déchiffrement.

On définit un *cryptosystème* par un procédé mathématique pour transformer d'une façon unique un message écrit en clair en un message dit *chiffré* afin qu'il soit inintelligible pour ceux à qui il n'est pas destiné [GUI-82].

On définit les termes suivants [KRA-87] :

- le *chiffrement* est l'opération qui consiste à transformer un texte clair en cryptogramme;
- le *cryptogramme* est le message écrit en caractères secrets, appelés *chiffres*;

- le *déchiffrement* est l'opération qui consiste à rétablir en clair un message chiffré, à l'aide de la clé;
- le *décryptage* est l'opération qui consiste à traduire des messages chiffrés dont on ne possède pas la clé;
- la *cryptanalyse* est l'art de décrypter les chiffres;
- la *cryptographie* est l'art de chiffrer et de déchiffrer des messages;
- la *cryptologie* est la science du chiffre dans ses deux aspects : cryptographie et cryptanalyse.

Une évaluation de la "solidité" d'un algorithme de chiffrement est donnée par sa capacité à résister efficacement aux attaques d'un cryptanalyste en possession de l'algorithme et d'un nombre illimité de couples (clair,cryptogramme), mais pas de la clé.

Un algorithme résiste efficacement à une cryptanalyse si la seule méthode connue pour découvrir la clé est de les essayer toutes les unes après les autres (*recherche exhaustive*). Un algorithme est dit *cassé* si on connaît une méthode rapide pour le cryptanalyser.

Enfin, on distingue généralement deux grandes catégories de cryptosystèmes [BAU-87] :

- les cryptosystèmes à clé secrète, ou *symétriques*, pour lesquels la même clé est utilisées lors du chiffrement et du déchiffrement;
- les cryptosystèmes à clé publique, ou *asymétriques*, pour lesquels la clé de chiffrement est publique, la clé de déchiffrement restant bien entendu secrète.

1.2. Les cryptosystèmes symétriques [DAV-84]

Ces cryptosystèmes se répartissent en trois grandes catégories qui sont les transpositions, les substitutions et les cryptosystèmes mixtes.

1.2.1. Les transpositions

Le cryptogramme est constitué de permutations des caractères du message en clair. Une transposition est parfois désignée par le mot *permutation*.

1.2.2. Les substitutions

Le cryptogramme est constitué par la traduction de chacun des caractères du message en clair en un autre alphabet.

1.2.3. Les méthodes mixtes

Les cryptosystèmes mixtes mélangent les transpositions et les substitutions. A l'heure actuelle, le système mixte le plus célèbre est le DES (Data Encryption Standard) [FIPS-77].

Le DES permet de chiffrer un bloc de 64 bits à l'aide d'une clé de 56 bits. Nous en donnons une présentation rapide ci-dessous (voir figure 5.1.) (1).

Une permutation fixe (I.P. : Initial Permutation), indépendante de la clé, est tout d'abord appliquée au bloc initial de 64 bits.

Le résultat de cette permutation est alors divisé en deux sous-blocs de 32 bits chacun : L (Left) et R (Right).

Seize étapes de chiffrement sont ensuite effectuées, dans lesquelles intervient la clé ...

... et à l'issue desquelles les deux blocs résultats L' et R'sont réunis pour former un nouveau bloc de 64 bits.

On applique à celui-ci une permutation fixe (F.P. : Final Permutation), indépendante de la clé, pour enfin trouver le cryptogramme.

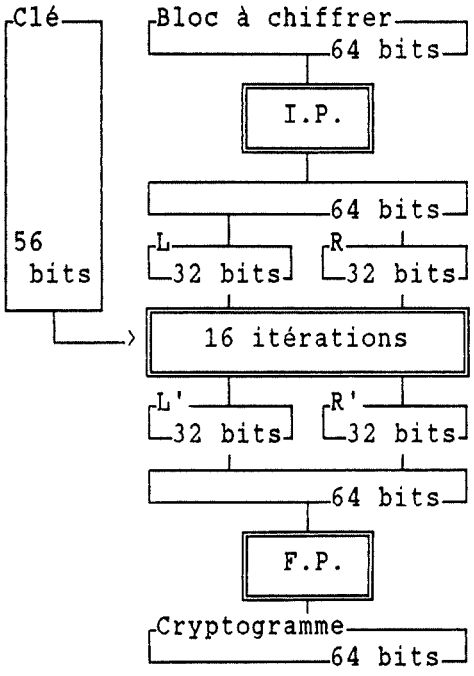

Figure 5.1. : Principe général du DES

L'itération i reçoit deux blocs de 32 bits : L_{i-1} et R_{i-1}. Son but est de les transformer en deux autres blocs de 32 bits : L_i et R_i, liés

1) Cette présentation est à notre sens plus claire que le diagramme figurant dans la publication originale. Nous la devons à monsieur LOUIS, de la firme CTI.

aux blocs d'origine par les formules (⊕ représente le OU exclusif bit à bit) :

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

La fonction f est composée d'une transposition initiale E, qui introduit de la redondance en transformant le bloc de 32 bits R_j en un bloc de 48 bits.

Ce bloc subit alors un OU exclusif bit à bit avec K_j qui est une transformée de la clé K, différente à chaque itération.

Le bloc de 48 bits ainsi obtenu est divisé en 8 "sixtets".

Chacun d'entre eux subit une substitution sélective, qui le transforme en un bloc de 4 bits.

Ces 8 blocs de 4 bits sont alors concaténés pour former un nouveau bloc de 32 bits, ...

... qui subit une permutation finale pour donner le résultat : R_{j+1}.

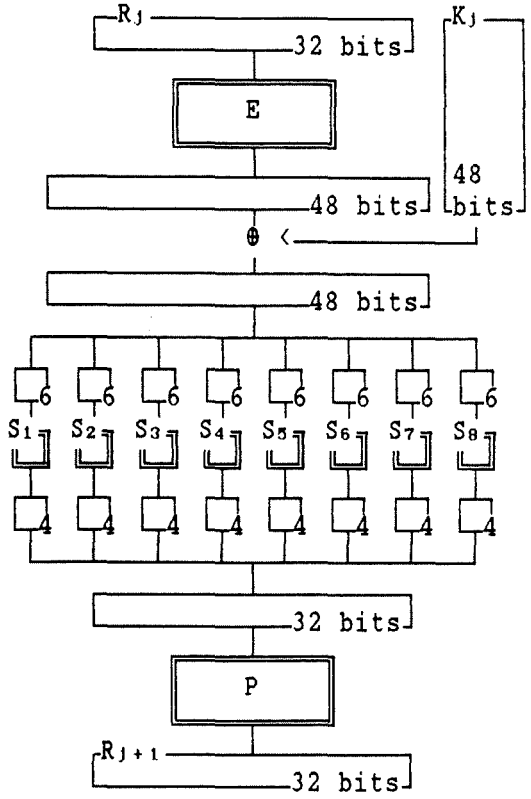

Figure 5.2. : La fonction f du DES

Le déchiffrement s'effectue de manière analogue. Les blocs R_{i-1} et L_{i-1} sont obtenus par les formules :

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(L_i, K_i)$$

Il n'est donc pas nécessaire d'inverser la fonction f, sur laquelle repose toute la sécurité du DES.

Notons ici que plusieurs spécialistes de la cryptographie (HELLMAN, par exemple) estiment possible, dans un proche avenir, la création d'une machine capable de tester en un temps réduit (une journée, par exemple) l'ensemble des clés possibles pour déterminer celle qui conduit à un cryptogramme donné. Une telle évolution de la technologie rendrait hasardeuse toute politique de sécurité basée sur l'utilisation de l'algorithme DES. Notons également qu'à ce jour il n'a pas encore été cassé.

1.3. Les cryptosystèmes asymétriques [DAV-84]

Introduits en 1976 par DIFFIE et HELLMAN [DIF-76], ces cryptosystèmes reposent sur la notion de fonction à trappe, c'est-à-dire une fonction dont la définition ne suffit pas à savoir en calculer rapidement l'inverse à moins de connaître une information supplémentaire connue de son concepteur : la trappe.

DIFFIE et HELLMAN définissent comme suit une fonction à trappe :

- Soient K un espace fini de clés, M un espace fini de messages et (P_k, S_k) des couples de transformation définis dans M.
- Pour toute clé k, P_k et S_k sont inverses;
- pour toute clé k et pour tout message m : $P_k(m)$ et $S_k(m)$ sont faciles à calculer;
- pour presque toutes les clés k, il est pratiquement impossible de définir un algorithme efficace équivalent à S_k en connaissant seulement P_k;
- pour toute clé k, il est facile de générer un couple (P_k, S_k).

Ainsi, lorsqu'un utilisateur désire être membre d'un cryptosystème à clé publique, il utilise un générateur public de chiffre à clé publique pour obtenir un couple (S,P). Il garde secrète la transformation S tandis qu'il inscrit la transformation P dans un annuaire public. Toute personne connaissant la transformation P peut envoyer un message m confidentiel à cet utilisateur en lui transmettant le cryptogramme C = P(m). Seul quelqu'un connaissant la transformation (secrète) S peut correctement déchiffrer C.

Actuellement, les deux fonctions mathématiques utilisées pour réaliser des systèmes de chiffrement asymétriques sont la fonction exponentielle sur un corps fini et la fonction puissance modulo n où n est le produit de deux grands nombres premiers tenus secrets.

L'algorithme RSA [RSA-78] est basé sur une fonction de la seconde catégorie; il présente le double avantage de jouir d'une description élégante et d'avoir résisté jusqu'à présent aux attaques de toutes sortes dont il a été l'objet [BAU-87]. Nous en présentons le principe ci-après à titre d'illustration.

Grossièrement, il est basé sur le fait qu'il est beaucoup plus facile de savoir si un grand nombre est composé (c'est-à-dire non premier)

que de déterminer les facteurs dont il est le produit.

L'utilisateur choisit donc deux grands nombres premiers p et q, puis calcule leur produit n. Il choisit ensuite un nombre e inférieur à n et rend public le couple (e,n) qui constitue sa clé publique. En revanche, il garde secrets p et q (la trappe); s'il les a choisis suffisamment grands, il est impossible à quiconque de les déduire de n.

A partir de p, q et e, l'utilisateur calcule ensuite un nombre d (sa clé secrète) tel qu'on ait :

$$de = ed = 1 \quad \text{(modulo n)}.$$

Moyennant une condition sur e, un tel d existe toujours; mais il est infaisable de le calculer sans connaître p et q.

Les fonctions P (publique) et S (secrète) sont alors définies sur les entiers modulo n par :

$$P(x) = x^e \quad \text{(modulo n)}$$
$$S(x) = x^d \quad \text{(modulo n)}.$$

On a donc, pour tout x ($0 \leq x < n$) :

$$S(P(x)) = (x^e)^d = x^{ed} = P(S(x)) = x^{de} = x^1 = x \quad \text{(modulo n)}$$

1.4. Conclusion

Deux philosophies différentes coexistent pour les cryptosystèmes. L'optique traditionnelle des cryptosystèmes symétriques impose aux deux correspondants (chiffreur et déchiffreur) de posséder la même clé, et de la tenir secrète, ce qui peut poser des problèmes de gestion des clés ("Key Management"). La seconde optique, par contre, autorise une plus grande souplesse : n'importe qui peut envoyer un message chiffré à un utilisateur répertorié, celui-ci sera le seul à pouvoir le déchiffrer en utilisant sa clé secrète.

Nous allons à présent examiner quelles techniques peuvent être utilisées pour sécuriser les logiciels.

2. Protection de la confidentialité du logiciel

Le possesseur d'un logiciel peut vouloir conserver l'exclusivité de la lecture du code de son logiciel; par conséquent, il verrouille aussi son droit d'exécution sur la copie qu'il détient.

Nous présentons ici deux techniques permettant de protéger la confidentialité d'un objet, et en particulier d'un logiciel. Ce sont le

chiffrement et l'utilisation d'un schéma à seuil.

2.1. Le chiffrement du code du logiciel

Le problème peut être résolu par des méthodes cryptographiques classiques : il suffit de stocker le logiciel en chiffrant son code, de manière à le rendre incompréhensible à toute personne ne possédant pas la clé de déchiffrement. Les différentes techniques de chiffrement ont été présentées à la section précédente; le choix de l'une d'entre elles (symétrique, asymétrique) se fera en fonction des besoins tant en sécurité qu'en performances, étant bien entendu que la méthode idéale n'existe pas !

Dans le cas du choix d'une solution ayant recours à des méthodes cryptographiques, le problème de la confidentialité du code du logiciel est reporté sur celui de la clé de déchiffrement; on suppose donc ici qu'il est plus facile de tenir secrète la clé plutôt qu'un long fichier, ce qui est une hypothèse raisonnable. Dans certains cas, on pourra recourir à des technologies spéciales pour garantir la confidentialité de la clé, comme par exemple son stockage dans une carte à microprocesseur (voir la section 6 du présent chapitre).

2.2. Fragmentation-dissémination et schémas à seuil [CAM-87, RAN-87]

La solution qui vient d'être exposée garantit la confidentialité de l'objet protégé, pour autant que la clé reste secrète. Celle-ci constitue donc un noyau dur pour la sécurité (si un cryptanalyste arrive à recalculer la clé, toute la protection est compromise).

Dans la cas où le système est très exposé à la malveillance, on désire minimiser les noyaux durs de sécurité tout en maximisant la disponibilité des objets archivés (c'est-à-dire minimiser la quantité de dégâts qu'une personne malveillante pourrait commettre dans un système qu'elle réussirait à pénétrer). Ces deux objectifs, à première vue contradictoires, peuvent être atteints en utilisant une technique originale appelée la *fragmentation-dissémination*, que nous présentons brièvement ici.

2.2.1. Principe

Cette technique, qui nécessite un environnement du type réseau (voir figure 5.3.), consiste à découper les informations sensibles en des parties élémentaires appelées *fragments* (fragmentation), puis à distribuer

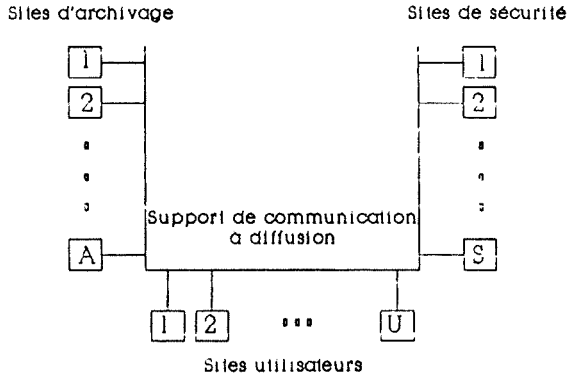

Figure 5.3. : L'environnement de la fragmentation-dissémination

les différents fragments sur des sites spécialisés (*sites d'archivage*) afin de les stocker en différents endroits (dissémination). C'est ce qui est expliqué à la figure 5.4.

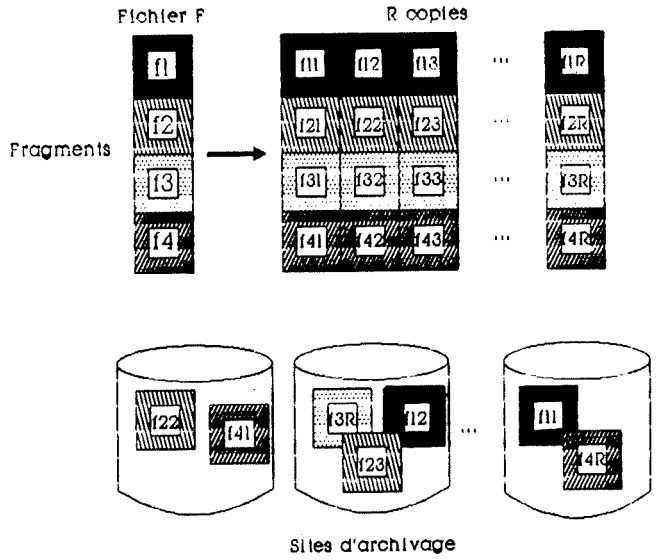

Figure 5.4. : Architecture du système

Il résulte de ce principe que la connaissance d'un ou de plusieurs fragments (mais pas tous) ne permet pas de reconstituer l'information originale. Un intrus accédant à divers sites ne pourrait obtenir tous les fragments que s'il contrôlait le système entièrement. Les fragments sont

de plus archivés en plusieurs exemplaires, ce qui assure la disponibilité en cas de destruction d'un ou de plusieurs sites.

2.2.2. La fragmentation

La fragmentation consiste donc à obtenir, à partir d'un objet de taille quelconque, des fragments qui ne délivrent une information significative que lorsqu'ils sont tous réunis dans le bon ordre. Ceci est réalisé à l'aide de techniques cryptographiques spéciales [FRA-86], utilisant une clé de fragmentation. Cette clé doit être protégée contre les lectures illicites.

La protection de ces clés de fragmentation est réalisée au sein des *sites de sécurité*. Une méthode cryptographique classique pourrait être utilisée pour protéger ceux-ci, mais cette méthode est contradictoire avec le principe de minimisation des noyaux durs de sécurité (si l'information d'un seul de ces sites était volée par un intrus, celui-ci posséderait alors la clé de fragmentation). Une autre solution peut être envisagée; elle consiste à cacher les informations des sites de sécurité par un schéma à seuil.

2.2.3. Schéma à seuil pour la protection des clés

Un schéma à seuil est une méthode qui permet de fractionner une information (par exemple une clé) en N images, de telle façon que la connaissance d'au moins S images (choisies au hasard) est nécessaire et suffisante pour reconstituer l'information (S est appelé le *seuil* et $1 \leq S \leq N$). Par conséquent, la connaissance d'au plus (S-1) images ne donne aucun complément d'information.

De ce fait, un intrus devra s'introduire dans au moins S sites de sécurité pour obtenir une information; de même, on peut perdre jusqu'à (N-S) images de l'information en gardant la faculté de pouvoir la reconstituer.

Un tel schéma peut être réalisé [CAM-87] en utilisant les codes détecteurs et correcteurs d'erreurs employés dans les télétransmissions, par exemple le code de REED-SOLOMON qui permet de corriger (2d+a) perturbations, où d est le nombre d'effacements (destruction d'un site) et a le nombre d'erreurs (altérations) dans un mot de code (l'ensemble des images).

La méthode d'interpolation polynomiale permet également de réaliser un schéma à seuil [RAN-87]. Elle utilise la propriété suivante : connaissant S couples (x_i, y_i), on peut trouver un polynôme unique H de degré (S-1) dont la valeur au point x_i est $y_i = H(x_i)$, quel que soit i. Il est de plus possible de générer plus de S couples (x_i, y_i) répondant au problème pour un polynôme H donné.

Un exemple trivial de cette méthode consiste à choisir S = 2. Connaissant 2 points du plan, on peut trouver une droite passant par ces deux points. Il est aisé de trouver d'autres points sur cette même droite. On obtient ainsi un schéma à seuil car n'importe quel couple de points permet de déterminer la droite alors que la connaissance d'un seul point ne donne aucune information puisque l'on peut faire passer une infinité de droites par ce point.

En résumé, la fragmentation-dissémination permet de minimiser les point durs de sécurité d'un système. L'utilisation d'un schéma à seuil permet de cacher la clé entre les différents sites de sécurité. De plus, l'utilisation de ces deux techniques combinées accroît la disponibilité de l'information en cas de destruction de quelques sites d'archivage ou de sécurité.

3. La protection du distributeur du logiciel [LOU-88, COO-84]

Les protections développées par les sociétés de conception de logiciels essaient d'endiguer le nombre de copies sauvages. La création d'un logiciel représente un investissement important qu'il convient d'amortir en commercialisant suffisamment d'exemplaires. La multiplication des copies non autorisées diminue les revenus légitimes des concepteurs pour qui seules les ventes constituent une source de recette.

Les différentes solutions doivent tenir compte de la nature de l'environnement pour lequel le logiciel a été créé. Par conséquent, le coût de la solution adoptée, qui est forcément limité par celui du logiciel lui-même, sera proportionnel à la valeur du matériel exécutant l'application.

Les solutions envisagées se présentent en général sous la forme d'un ensemble d'instructions noyées dans le code de base. Celles-ci sont chargées d'exécuter un contrôle pour vérifier que le logiciel est bien un exemplaire fourni par la société qui le commercialise.

3.1. Contrôle de l'existence d'un droit

3.1.1. Principe

La société qui distribue le logiciel va l'apparier à un numéro de machine. cette solution est bien adaptée pour les gros systèmes. des techniciens de maintenance viennent installer le logiciel sur la machine qui va l'exécuter. Le client paie un droit d'utilisation du logiciel et éventuellement passe un contrat de maintenance.

L'adéquation la plus performante est réalisée lorsque le constructeur de la machine est aussi le créateur ou le distributeur exclusif du logiciel. Il peut utiliser toutes les subtilités de son matériel pour protéger l'utilisation de son logiciel. Il peut également profiter des journées de maintenance pour désamorcer des "bombes logiques" disséminées dans les logiciels et destinées à altérer son comportement si le client n'a pas renouvelé son contrat. Ces modifications du comportement peuvent être activées par comparaison à des dates de validité ou à des valeurs contenues dans des zones de données (2).

Pour certains logiciels très spécialisés comme les systèmes de C.A.O., MENTOR GRAPHICS, qui fonctionnent sur les stations APOLLO, le fabriquant envoie par courrier des mots de travail d'une durée limitée, que le responsable enregistre dans un fichier prévu à cet effet. Le système dispose d'une horloge interne et vérifie la date de péremption et la validité du mot de travail dans le fichier. Le système compare, lorsqu'il accède à un fichier, les dates de création et de dernier accès avec la date courante du système et refuse l'accès au fichier en cas d'incohérence. Ainsi, si l'utilisateur a modifié la date et l'heure courante du système dans le but de frauder, il ne pourra plus récupérer ses fichiers.

Dans le cadre d'une utilisation sur des systèmes mini ou micro, les distributeurs construisent des cartes électroniques à intégrer dans la machine et qui porteront un identificateur ou assureront des opérations spécifiques telles que l'émission de réponses temporisées sur présentation de valeurs aléatoires. Le principe est d'obtenir une réponse du dispositif interne qui soit fonction de la requête émise et différente à chaque

---

2) C'est un bel exemple de cheval de Troie à la frontière entre le bon droit et la malveillance - voir à ce sujet la section consacrée à la malveillance contre les systèmes (première partie, chapitre 2).

vérification. On peut tester la valeur reçue ou mesurer le temps nécessaire à l'émission de la réponse. Ces méthodes peuvent faire appel à de véritables fonctions électroniques câblées ou plus simplement à des matrices de codage enregistrées dans des mémoires mortes.

3.1.2. Limites de la méthode

La robustesse de ces solutions tient dans la manière de noyer le code de vérification dans l'ensemble du logiciel. De ce fait, plus la machine est importante et le logiciel complexe, plus la localisation de la routine est ardue.

Les méthodes de mots de passe et de dates de péremption sont donc bien adaptées aux ordinateurs centraux, ainsi qu'aux logiciels spécialisés qui tournent sur des machines telles que les stations de travail de CAO (3). Elles ne sont pas très difficiles à mettre en oeuvre si on prend soin de les concevoir dès le début de l'analyse de conception du logiciel. Par contre, la gestion des mots de passe saisonniers peut s'avérer lourde à mettre en place chez le distributeur et relativement coûteuse. On réservera cette méthode à des logiciels loués.

Le coût des cartes additives pour mini ou micros n'est pas négligeable, alors que la fiabilité du système repose sur la possibilité de retrouver la partie du logiciel qui effectue la vérification. Elles sont également espionnables au moyen d'un analyseur logique.

Elles présentent l'inconvénient d'utiliser un emplacement physique dans le fond du panier de la machine, qui aurait pu servir à l'installation d'une carte ETHERNET, par exemple. Il n'est donc pas possible d'étendre cette technique tant que les constructeurs ne se mettent pas d'accord sur un modèle précis. Sans cela, la totalité des emplacements physiques de fond de panier des micro-ordinateurs est monopolisée par les systèmes de sécurité (qui, rappelons-le, sont imposés à l'utilisateur par le constructeur ...). Leur emploi est limité et peu pratique pour l'utilisateur, qui a par contre la possibilité d'effectuer librement des copies de sauvegarde.

Ce type de protection est applicable pour des micro-ordinateurs isolés ou banalisés qui n'exécutent que des logiciels d'application dans un environnement aux fonctions système limitées. Il ne faut pas que ces

---

3) Conception Assistée par Ordinateur

applications soient trop nombreuses pour les raisons évoquées précédemment. Elles ne résistent pas à des informaticiens professionnels.

3.2. Contrôle sur la disquette

La méthode la plus couramment utilisée par les distributeurs de logiciels pour micro-ordinateurs consiste à faire porter par le support lui-même la preuve qu'il est celui qui a été légalement distribué. De fait, les logiciels pour micro-ordinateurs sont commercialisés sous forme de disquettes; ce sont elles qui vont véhiculer la protection. Le concepteur de cette protection utilise toutes les caractéristiques techniques des disquettes, de leur contrôleur et de leur lecteur pour essayer de marquer de manière indélébile et incopiable une sorte de certificat de provenance [HERB-84].

3.2.1. Rappels sur la structure des disquettes

Les disquettes sont les supports magnétiques d'information privilégiés pour les micro-ordinateurs [HERB-84].

Une disquette est découpée en un ensemble de *pistes* concentriques dont la position par rapport au centre de la disquette est déterminée par le déplacement du bras qui supporte la tête de lecture. Les pistes sont composées de bits enregistrés sur la circonférence.

Pour faciliter la manipulation des informations, on procède au découpage logique des pistes en *secteurs*. Un secteur est la plus petite quantité d'information accessible sur la disquette. L'emplacement du premier secteur de chaque piste est aligné sur un trou physique percé dans la disquette et qui sert d'index. Les autres secteurs sont retrouvés grâce à des en-têtes enregistrées dans le cas du formatage logiciel ou par des trous physiques dans le cas du formatage matériel (4). Les secteurs d'une piste sont séparés par des "GAP's", espaces réservés destinés entre

4) Cette dernière méthode est actuellement beaucoup moins utilisée.

autres à la synchronisation.

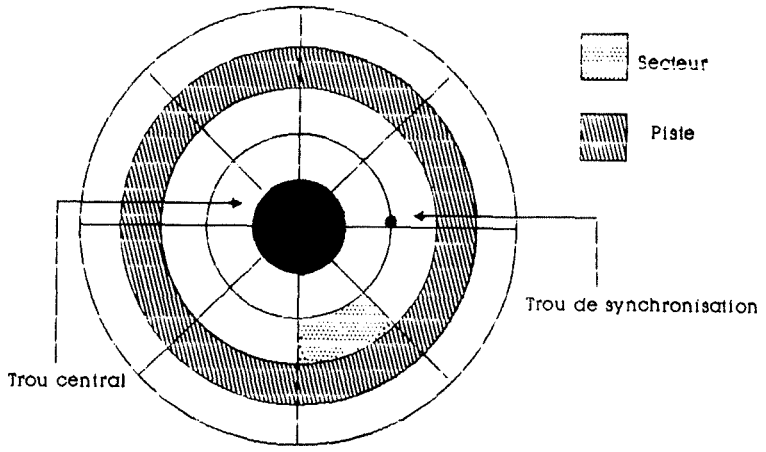

Figure 5.5. : Organisation d'une disquette

3.2.2. Protection relative au secteur

Dans cette catégorie de protection, le concepteur joue sur les caractéristiques des secteurs. Le but est de rendre un élément du secteur inaccessible par une lecture normale. Le concepteur de la protection peut agir sur l'en-tête du secteur ou sur celui des données. Une solution consiste à créer un faux en-tête de secteur pour leurrer le contrôleur (les informations lues ne correspondent pas au format usuel). La génération d'un code de détection d'erreur erroné provoque également une erreur de lecture [MAC-87]. Enfin on peut introduire des secteurs de longueurs différentes lors du formatage, ce qui rend leur accès compliqué et impossible sans manipulation des caractéristiques du format.

3.2.3. Utilisation de paramètres temporels

Cette catégorie de protection utilise des paramètres temporels liés à la vitesse de rotation de la disquette ainsi qu'à celle nécessaire au déplacement de la tête de lecture.

Connaissant ces caractéristiques temporelles, le concepteur de la protection peut essayer d'organiser la présence des secteurs sur les pistes de sorte qu'il puisse, en les lisant, savoir si c'est lui qui a créé la disquette ou s'il s'agit d'une copie avec formatage standard.

Une technique couramment utilisée pour accélérer l'accès au secteur peut être reprise pour marquer la disquette. Cette méthode est l'entrela-

cement. Les secteurs ne sont pas écrits de façon contiguë par numéro croissant mais décalés les uns des autres pour tenir compte du temps de lecture et de la vitesse de rotation de la disquette. Ainsi, pendant le temps de la prise en compte de la lecture du premier secteur, la disquette aura tourné mais la tête de lecture sera sur le début du secteur numéro deux. On accélère la vitesse des lectures séquentielles en évitant d'attendre une rotation complète de la disquette.

Pour marquer la disquette selon cette méthode, il suffit de savoir que si l'on attend un temps fixé après la lecture d'un secteur déterminé, la tête de lecture doit être en regard d'un secteur au numéro prévu. On peut également demander un déplacement de la tête vers une autre piste et regarder quel est le numéro du secteur immédiatement disponible sous la tête de lecture.

La technique de temporisation peut être efficace si on a créé des GAPs élargis entre les secteurs des pistes. Après s'être synchronisé sur un numéro de secteur choisi, on lance une temporisation avant de demander une lecture. Pour une vitesse de rotation fixe, le numéro du secteur lu sera différent selon que les GAPs inter-secteurs ont été élargis ou non. La solution des GAPs élargis permet parfois d'y introduire de petits messages dont on pourra contrôler la présence ou le séquencement.

3.2.4. Protection relative aux pistes

Dans cette série de marquage, la principale entité manipulée est la piste. La première idée consiste à ajouter une piste supplémentaire par rapport au standard de formatage, et dont les secteurs contiennent des informations de contrôle.

Les pistes concentriques sont créées sur la disquette à n'importe quel endroit du support. Il n'existe pas de marques physiques qui positionnent latéralement les pistes les unes par rapport aux autres. Cette caractéristique est fonction de la technique de déplacement du bras qui peut être un moteur linéaire ou pas à pas. Sur un moteur pas à pas, c'est le nombre d'impulsions envoyées au moteur qui commande le bras alors que les moteurs linéaires déterminent si la tête est bien centrée sur une piste en mesurant le plus fort signal reçu. L'emplacement des pistes est fixé pour un type de machine et peut être utilisé pour marquer les disquettes, si la logique de commande des moteurs est accessible.

3.2.5. Autres types de protection

Pour assurer la protection de ses logiciels, le concepteur peut décider d'utiliser une technique de codage des informations différente de celle en vigueur sur la machine.

On peut également générer une mauvaise piste ou un mauvais secteur sur la disquette en se servant d'un laser. Le concepteur de la protection pratique un minuscule trou au laser, détériorant le support. Le trou reste invisible à l'oeil et n'est pas reproductible. Le programme de protection vérifiera que le secteur ou la piste ainsi marqués ne sont pas lisibles.

3.2.6. Limites de la méthode

Les méthodes de protection par disquette sont aujourd'hui les plus répandues dans le domaine de la micro-informatique. Elles ne sont pas coûteuses puisqu'elles ne nécessitent aucun apport particulier. En effet, les disquettes restent le support de vente privilégié des logiciels micro. Elles présentent par contre l'inconvénient majeur de ne plus permettre à l'utilisateur d'effectuer de copies de sauvegarde.

Devant le désarroi des utilisateur, des sociétés ont développé des logiciels spécialisés qui se chargent de recopier l'image binaire d'une disquette source vers une disquette cible. Ces logiciels sont capables de déjouer les principales méthodes de conception. Les journaux spécialisés dans la micro-informatique comportent des publicités pour la famille des logiciels "copywrite". Ils sont disponibles à très bas prix et ne sont naturellement pas protégés. On les retrouve donc dans le répertoire réservé aux utilitaires de la plupart des compatibles P.C. Les méthodes de protection utilisant la disquette ne sont dès lors pas efficaces pour ce type de configuration. De plus, il faut prévoir une deuxième méthode de protection pour le logiciel installé sur le disque dur.

3.3. Protection par programme de lancement

3.3.1. Principe

Le principe de l'installation du logiciel sur le disque dur est de recourir à un programme de lancement ou *d'installation*. Ce programme est

capable de fonctionner une ou plusieurs fois en fonction de la politique commerciale suivie par le constructeur. Il dispose pour cela d'un compteur d'installation sur la disquette ou s'autodétruit après avoir chargé le logiciel sur le disque. Afin de ne pas permettre à l'utilisateur de charger lui-même le logiciel, les instructions de celui-ci sont déstructurées par rapport à un fichier classique. On peut utiliser la méthode de déstructuration sur la disquette et employer un programme lanceur pour l'exécution du logiciel, mais les logiciels professionnels fonctionnent généralement sur disque dur.

Une fois installé sur le disque dur, le logiciel n'est plus protégé par la même méthode que celle qui était utilisée sur la disquette. Le moyen le plus simple de vérifier qu'il n'est pas une copie pirate consiste à demander la présence dans le lecteur la disquette originale; ce contrôle doit être réalisé en permanence.

Le deuxième axe de protection possible est l'inscription par le programme de chargement d'indications relatives à sa position physique et logique sur le disque dans un espace de données réservé. Une routine de vérification interne au logiciel ira régulièrement tester cette valeur et la comparera à sa position courante.

3.3.2. Limites de la méthode

Ces protections sont faciles à mettre en oeuvre et ne coûtent rien. Elles ne seront efficaces que si la protection sur la disquette de distribution est robuste. En effet, à partir du moment où l'on peut dupliquer cette disquette, on est capable de faire autant d'installations sur disque dur qu'on le désire.

S'il y a plusieurs utilisateurs pour le micro-ordinateur, le contrôle de la présence de la disquette originale n'est pas possible à utiliser. Par contre, l'autre catégorie de protection (contrôle sur le disque dur) s'applique pour l'utilisation de micros en libre-service ou banalisés. On protège les logiciels de la duplication par les utilisateurs de la machine, tout en sachant que le responsable a pu faire autant de copies de la disquette originale qu'il a voulu. Ces méthodes ne sont donc pas adaptées pour les machines isolées ou personnelles.

3.4. Contrôle par un élément externe

Le défaut principal des protections sur disquette est qu'elles ne permettent pas de faire de copies de sauvegarde. Pour remédier à ce handicap, on associe la disquette d'origine à un dispositif matériel ou "bouchon". Ce dispositif se branche sur un des ports de la machine.

3.4.1. Principe

La plupart des micro-ordinateurs disposent d'une carte de communication série. Le bouchon reconnaîtra qu'il est sollicité quand il y détectera une séquence spéciale. Il émettra alors une réponse sous la forme d'une séquence reprenant des caractéristiques identiques à celles qu'il avait reconnues. Dans d'autres cas, le bouchon se connecte sur un port parallèle.

On retrouve ces produits commercialisés sous différents noms par plusieurs sociétés, comme :

- DONGLE PROGRAMMABLE, par ELECTRYON,
- TOUR DE BABEL, par XPRES,
- CLE ELECTRONIQUE, par MICROPHAR.

3.4.2. Limites de la méthode

Les prix de ces composants sont élevés en regard des possibilités de localisation de la routine de vérification sur les micro-ordinateurs. Les trappes d'interruptions et les debuggers sont alors des outils efficaces et plus faciles à se procurer qu'un microscope électronique ou qu'un analyseur logique pour espionner la partie matérielle.

Cette faiblesse, alliée à la difficulté d'empiler mécaniquement les bouchons les uns sur les autres, fait qu'ils ne sont pas adaptés à des logiciels bon marchés.

L'utilisation sur micro-ordinateur isolé est possible. Par contre, on se heurte à un problème de distribution dès qu'il y a plusieurs utilisateurs car il faut donner à chacun d'eux autant de dispositifs que de logiciels utilisés, ce qui représente parfois un trop grand nombre de bouchons.

3.5. Protection par le mode de distribution

Il convient de relativiser quelque peu l'impact des copies pirates sur la distribution des logiciels. S'il est certain que cela pose un problème en ce qui concerne les jeux et l'informatique domestique, ce n'en est pas un pour l'informatique professionnelle. Les entreprises ont d'autres préoccupations que celles de se livrer au sport particulier du "déplombage". Elles exigent des logiciels fiables et un service après-vente irréprochable. Actuellement, on ne conçoit pas de fournir un logiciel à un employé sans formation ni documentation; l'investissement que représente l'acquisition d'un logiciel doit être rentabilisé par une utilisation optimale de ses fonctionnalités. La démarche est donc diamétralement opposée à celle des utilisateurs d'ordinateurs personnels.

3.5.1. Principe

La politique de distribution du logiciel peut être un facteur de protection. Certains fabricants lient leurs produits à un numéro d'identification, qui est enregistré dans le fichier des clients en correspondance avec la commande passée. Lors d'une intervention chez le client ou lors d'un appel téléphonique de sa part, il est possible de vérifier qu'il est bien le possesseur légitime du logiciel en comparant le numéro d'identification à celui qui se trouve dans le fichier des commandes relatives à ce client. Cette méthode est simple, puisqu'elle repose sur des informations déjà saisie et mises à jour dans le cadre de la gestion des ventes.

Enfin, certains logiciels se prêtent bien à la protection. C'est le cas des progiciels de télécommunication, par exemple. Ils peuvent être développés de deux manières :

- soit en écrivant des programmes autour d'un matériel d'adaptation au medium conçu et réalisé par une autre société, et disponible séparément sur le marché;
- soit en envisageant l'étude comme un tout, en construisant un adaptateur spécifique auquel on confiera certaines fonctionnalités du logiciel.

Dans le deuxième cas, la distribution du logiciel sera indissociable de celle de l'adaptateur. Le logiciel est donc protégé puisque, pour pouvoir en utiliser une copie, il faut copier également le matériel ... ce qui est

probablement plus coûteux que de l'acheter !

3.5.2. Limites de la méthode

La technique de développement sur un matériel spécifique est séduisante mais se limite à des domaines d'activité très restreints. De plus, les sociétés qui conçoivent les logiciels ne sont pas toujours capables de créer le matériel associé. Ce type de protection s'adapte par contre à tous les cas d'utilisation.

4. La protection de l'intégrité du logiciel

Ces protections ont pour but de contrer la malveillance dirigée contre l'intégrité du logiciel d'un utilisateur ([5]). Dans la suite, l'intégrité d'un logiciel désignera sa bonne conformité par rapport à sa version d'origine (en provenance du concepteur).

Deux types de parades existent; le premier consiste à protéger le logiciel contre toute modification, quelle qu'elle soit (par extension, on peut également le protéger contre des attaques visant à localiser des routines de vérification comme celles protégeant les droits du concepteur). Le deuxième type permet de détecter qu'un logiciel a été modifié par rapport à sa version originale.

4.1. Immuabilité du logiciel

La seule manière de rendre un logiciel irréversiblement non modifiable est de le stocker sur un support non réinscriptible (à condition bien entendu que seule cette copie soit utilisée).

Nous donnons ici les grandes caractéristiques de trois supports disponibles, dont tous possèdent l'avantage d'être (théoriquement) non modifiables :

- la ROM (Read Only Memory) est utilisées par les fabricants de composants informatiques pour stocker des programmes figés. Elle est programmée de manière irréversible par fusion sélective de liai-

---

[5]) Les méthodes spécialement conçues dans le but de se protéger des virus informatiques seront exposées au chapitre 3.

sons [MEI-84];

- le CD-ROM (Compact Disk-Read Only Memory) est le frère jumeau du disque compact audio. Son gros inconvénient est d'avoir un mode d'écriture par pressage d'un "master" par le constructeur. Son coût doit donc être amorti par la vente de nombreux exemplaires [ANO-86];
- le WORM (Write Once and Read Memory) est un disque optique numérique ineffaçable, offrant l'avantage de permettre directement à l'utilisateur d'écrire une fois pour toutes sur un disque vierge à l'origine [ANO-86].

Notons que ce type de protection n'est pleinement efficace que si l'accès aux supports est très soigneusement réglementé. En effet, une ROM peut être lue et son contenu analysé. Il est donc possible d'en réaliser une version frauduleuse qui pourrait remplacer l'ancienne [DAV-84].

4.2. Contrôle de l'intégrité du logiciel

La mise en place de moyens destinés à protéger l'intégrité des logiciels peut se faire à deux niveaux. Le premier sera qualifié de *préventif* et utilise les moyens de sécurité disponibles dans le système d'exploitation. Il s'agit de compliquer l'accès aux ressources informationnelles, ou les liens inter-processus qui garantissent le partage équitable de la machine.

Le second niveau consiste à élaborer des méthodes de *détection* sur le code généré. Elles exploiteront autant que possible leur environnement (capacités matérielles) mais aussi les outils du système d'exploitation de la machine.

4.2.1. Utilisation de la protection du système d'exploitation

4.2.1.1. Principe [LOU-88]

La majorité des systèmes d'exploitation multitâches ont été développés avec un souci de pouvoir protéger les utilisateurs entre eux, le système des utilisateurs, les utilisateurs contre le système et les processus d'un même utilisateur entre eux. Il existe donc des mécanismes chargés de contrôler l'accès aux ressources.

Un système d'exploitation contient de nombreuses ressources qui doivent être protégées. Elles peuvent être de nature matérielle comme le CPU ou la

mémoire, ou logicielle comme les fichiers ou les processus. La nature de chacune de ces ressources permet de dresser la liste des opérations qu'on peut lui appliquer. Par exemple, la lecture et l'écriture sont deux opérations de base exécutables pour un fichier.

On est alors capable d'établir une matrice des droits, dont les lignes désignent les permissions d'action d'un utilisateur sur l'ensemble des ressources qui forment les colonnes.

Ressources / Utilisateurs	Fichier f_1	Fichier f_2	Processus p_1
Processus 1	lire écrire	interdit	activer
Processus 2	lire	propriétaire	bloquer
Processus 3	écrire	XXXXXXXX	détruire

Figure 5.6. : Matrice des droits

Le problème posé dans le cadre du contrôle d'intégrité du logiciel est de limiter au maximum les ressources accessibles aux processus et de s'assurer qu'ils n'essaient pas d'en utiliser d'autres. Dans le cas où l'on constate qu'un processus demande à utiliser des ressources auxquelles il n'a pas accès, on peut supposer qu'il a été détourné de sa fonction originale et qu'il a donc été modifié.

Le contrôle d'intégrité d'un logiciel nécessite une bonne protection de la mémoire; c'est réalisé en définissant, pour chaque unité élémentaire d'allocation, quels sont les processus qui sont autorisés à les utiliser. De même, il faut s'assurer qu'un logiciel n'exécute pas d'instruction provenant d'un espace mémoire auquel il n'a pas accès et vérifier qu'il est de plus habilité à utiliser cette instruction.

En effet, certaines instructions peuvent être réservées au système, car elles permettent par exemple la mise en oeuvre des protections. Plus simplement, on peut souhaiter réaliser une hiérarchie sur plusieurs niveaux des instructions et des primitives du système. Certaines instructions sont

à surveiller parce qu'elles conditionnent un changement de pouvoir du processus; c'est le cas typique des instructions d'appel de sous-programme qui référencent parfois des primitives du système telles que les ouvertures de fichiers ou les écritures sur disque. Il faut adapter le droit associé au processus en fonction des entités manipulées, ce qui correspond par exemple au passage en mode super utilisateur ("super user" sous le système UNIX). Il ne faudra pas omettre de rétablir les anciens pouvoirs du processus au retour de la procédure.

4.2.1.2. Limites de la méthode

Le système d'exploitation peut être conçu de manière à protéger l'intégrité des logiciels. En fait, il s'agit surtout de détecter une modification des fonctions initiales du logiciel, ou de protéger les méthodes qui le sécurisent contre le piratage.

Cette méthode est adaptée aux systèmes d'exploitation multitâches; elle n'est pas applicable à un environnement micro-ordinateur, par exemple.

4.2.2. Contrôle sur les instructions

4.2.2.1. Principe

Pour vérifier qu'un logiciel n'a pas été modifié, on peut imaginer de contrôler les instructions qui le composent. La première méthode consiste à comparer toutes les instructions en mémoire à celles d'une version dont le système de protection serait certain de la validité. Cette solution n'est pas vraisemblable en utilisation continue mais elle pourrait être utilisée lors de visites de contrôle par une autorité supérieure. Toutefois, la vérification complète de tous les logiciels en cours d'exécution sur une machine serait très longue et nécessiterait l'arrêt automatique des traitements sans que l'utilisateur puisse dissimuler son forfait (6).

Lorsque le facteur temps est important, on peut alors imaginer un autre type de contrôle.

---

6) Une solution serait de faire exécuter le processus de contrôle dans un mode de fonctionnement à très forte priorité, provoquant la réquisition immédiate du processeur.

On peut ainsi calculer la valeur d'une fonction mathématique à partir de morceaux du logiciel et le stocker sur un support sécurisé. Lors du contrôle, le processus de vérification recalculera la même fonction sur les instructions du programme en attente et pourra détecter d'éventuelles modifications si le résultat obtenu est différent du résultat calculé au départ [POZ-86].

4.2.2.2. Quelles fonctions utiliser ?

La mise en oeuvre de la protection que nous venons de décrire passe par la définition de la fonction mathématique à utiliser.

Une première idée serait de réutiliser les fonctions employées en télécommunications [MAC-87]. Mais ces fonctions sont conçues dans le but de se protéger contre les erreurs aléatoires; de ce fait elles sont très vulnérables à des attaques volontaires menées par une intelligence malveillante.

Une seconde idée est alors d'utiliser les ressources offertes par la cryptographie, et plus particulièrement la notion de *code d'authentification* [GIL-88]. Cette notion est directement liée à celle de *fonction de condensation à sens unique*, que nous définissons ci-dessous [AKL-83] :

> Une fonction est dite *de condensation* si elle associe à un fichier de b bits (b quelconque) une et une seule valeur de m bits (m fixé et significativement plus petit que b). Une fonction H de condensation est dite *à sens unique* si :
>
> - pour n'importe quelle argument F appartenant au domaine de H, il est facile de calculer la valeur correspondante v = H(f);
> - étant donnés H et F, il est infaisable en pratique de trouver F' ≠ F de telle sorte que H(F') = H(F).

Le problème du choix de H est loin d'être trivial. Beaucoup de fonctions ont été proposées dans la littérature, dont un grand nombre ont été cassées depuis leur publication. A titre d'illustration, nous présentons l'une des plus simples d'entre elles, basée (comme la majorité des autres) sur l'utilisation du DES (7). Nous en proposerons ensuite une autre qui n'a

---

7) La plupart des fonctions proposées sont basées sur le DES. L'idée est de découper le fichier à vérifier en blocs de longueur fixe; c'est dans la manière de combiner ces blocs avec d'autres valeurs et avec des chiffrements que se trouve tout l'art du cryptographe.

pas encore été cassée à l'heure actuelle.

4.2.2.2.1. Le mode CBC défini pour le DES [FIPS-80]

Le fichier F est tout d'abord divisé en n blocs de 64 bits chacun appelés F_1, F_2, ... , F_n. Si le dernier bloc comporte moins de 64 bits, il est complété par des zéros.

La figure 5.7. illustre le fonctionnement de ce mode d'opération. Toutes les opérations travaillent sur des opérandes de 64 bits.

- Un bloc du fichier : F_i est combiné au résultat intermédiaire précédent (H_{i-1}) par un OU exclusif bit-à-bit. Elle a pour conséquence de lier au traitement du bloc F_i le résultat du traitement de tous les blocs qui l'ont précédé;
- le résultat obtenu est alors chiffré par le DES;
- le résultat du chiffrement, H_i, constitue le nouveau résultat intermédiaire qui sera utilisé lors du traitement du bloc suivant.

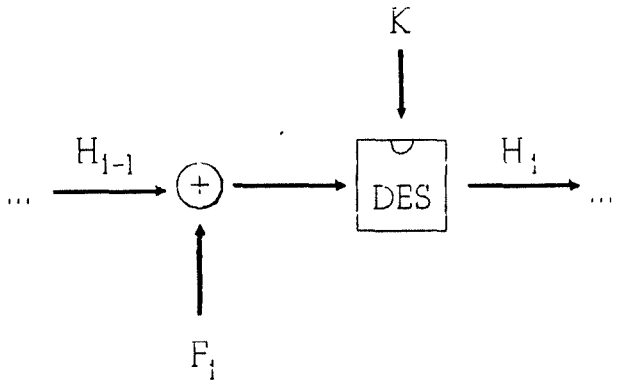

Figure 5.7. : Le mode CBC du DES

Une valeur d'initialisation est nécessaire car le premier bloc ne possède pas de "résultat intermédiaire précédent". Le code d'authentification du fichier est constitué par le dernier résultat intermédiaire H_n, ou par une partie de celui-ci.

4.2.2.2.2. Une fonction due à QUISQUATER

Nous n'entrerons pas dans les détails de cette fonction très compliquée proposée par QUISQUATER [GIL-88]. Le schéma global est le suivant :

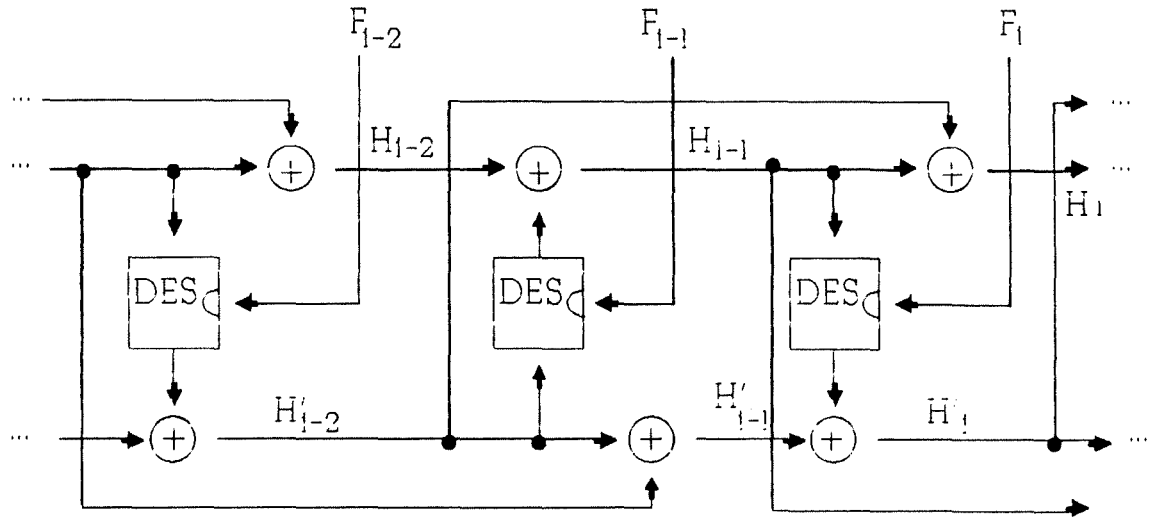

Figure 5.8. : fonction de QUISQUATER

Le code d'authentification final est obtenu par concaténation des résultats finaux H_n et H'_n.

4.2.2.3. Limites de la méthode

Notons tout d'abord que ces méthodes sont inefficaces si le fraudeur peut recalculer le code d'authentification et s'il peut en remplacer l'ancienne version par une version de son cru. Le système de protection intervient donc à deux niveaux : confidentialité de la clé et/ou de l'algorithme, et intégrité des codes d'authentification.

Le facteur temps est également une variable à prendre en considération. Ainsi, l'implantation de telles méthodes implique un découpage judicieux des logiciels en unités de base de calcul. La plus petite quantité allouée par le gestionnaire de la mémoire semble idéale pour servir d'unité de fractionnement si l'on opte pour une solution en temps réel [LOU-88]. La présence d'un dispositif matériel, assurant le calcul automatiquement pour chaque chargement de pages en mémoire, évite de trop perdre de temps dans les contrôles. Ce module peut également être doté de caractéristiques telles

qu'il en devienne virtuellement inviolable; on pourra alors l'utiliser pour y stocker des valeurs employées par les algorithmes (clés, ...) ainsi que les codes d'authentification eux-mêmes. Les trois limites du temps, de la confidentialité des clés et de l'intégrité des codes d'authentification sont donc ainsi partiellement résolues par l'utilisation d'un tel dispositif matériel. Il n'en reste pas moins que ces techniques sont lourdes à mettre en oeuvre sur l'ensemble des processus. Elles restent néanmoins efficaces pour vérifier l'intégrité des certaines primitives du système qui participent activement à la sécurité. C'est le cas par exemple des procédures de contrôle d'accès aux gros ordinateurs ("LOGIN").

4.2.3. Protections spécifiques contre les virus informatiques

Nous avons montré dans la première partie, chapitre 3 (page 25), que les virus constituent une grave menace pour les logiciels et pour les systèmes. Des protections existent pour s'en protéger avec certitude; d'autres permettent de limiter l'infection au sein d'un système contaminé.

Ces moyens de protection seront examinés en détail dans la seconde partie, chapitre 3 (page 94 et suivantes).

4.3. Conclusion

La protection de l'intégrité d'un logiciel peut être réalisée de différentes manières. La plus simple est probablement de le rendre immuable, mais ceci ne peut s'appliquer à tous les cas. Il convient alors d'utiliser une forme de contrôle de l'intégrité du logiciel, à l'aide de mécanismes offerts par le système d'exploitation pour tenter de prévenir les modifications non autorisées, ou par vérification des instructions du logiciel pour tenter de détecter d'éventuelles altérations.

5. Contrôle de l'accès au logiciel

Après avoir montré comment on pouvait prévenir les lectures, les copies et les altérations illicites d'un logiciel, nous allons envisager les moyens couramment utilisés pour en protéger l'accès.

Historiquement, ce sont ces techniques qui ont constitué les premières protection des logiciels. Le système d'exploitation vérifiait que tout

utilisateur demandant l'accès à une ressource (y compris à un programme) en avait bien le droit.

Cette manière de procéder est simple à mettre en oeuvre, naturelle, et assez efficace en pratique pour aider la plupart des indiscrets à résister à la tentation. C'est pourquoi la plupart des systèmes d'exploitation multi-tâches proposent encore à l'heure actuelle une protection des utilisateurs les uns des autres. Dans certains cas, le logiciel prendra en outre lui-même en charge son contrôle d'accès.

5.1. Protection d'accès par le système d'exploitation

Ces techniques de protection reposent sur le postulat suivant :

Le système d'exploitation a correctement identifié chaque utilisateur [ESS-86].

Nous examinerons donc quelques techniques d'authentification avant d'exposer la protection proprement dite offerte par le système d'exploitation pour l'accès aux logiciels.

5.1.1. Quelques méthodes d'identification

Le système d'exploitation se base sur la reconnaissance de quelque chose que l'utilisateur connaît, qu'il possède ou sur une de ses caractéristiques physiques pour pouvoir l'identifier.

5.1.1.1. Les mots de passe

5.1.1.1.1. Principe

La méthode la plus courante de vérification des "connaissances" de l'utilisateur est la présentation d'un couple (identificateur,mot de passe). Le programme de "LOGIN" demande à l'utilisateur de taper au clavier les caractères correspondant à son nom dans le système et ceux de son mot de passe. Celui-ci est immédiatement chiffré (sous UNIX, l'algorithme utilisé est le DES). Le programme recherche dans la table des mots de passe chiffrés une entrée correspondant au nom de l'utilisateur. S'il la trouve, la compa-

raison entre le mot de passe stocké et le mot de passe calculé conditionne l'entrée ou le rejet du système [VAX-85].

La forme extrême d'utilisation de mots de passe est le recours à un questionnaire aléatoire personnalisé auquel un et un seul utilisateur sera capable de répondre sans faire d'erreur.

Un intrus ne doit pas pouvoir lire la table des mots de passe, ni partiellement ni en totalité. C'est pourquoi les mots de passe y sont conservés sous forme chiffrée [ESS-86]. Ce chiffrement immédiat a également comme avantage de rendre impossible le vol du mot de passe par simple écoute sur la liaison terminal-ordinateur.

5.1.1.1.2. Limites de la méthode

Cette méthode n'est pas très fiable si on ne prend pas au sérieux le choix du mot de passe [ESS-86]. On peut attaquer le système en tentant une recherche exhaustive de tous les mots de passe à partir d'un terminal intelligent ou en orientant cette recherche par la méthode des mots habituels (8). On peut également écouter sur la liaison terminal-ordinateur pour connaître le mot de passe chiffré, que l'on n'a plus qu'à réinjecter sur la ligne comme s'il venait d'être introduit par l'utilisateur puis chiffré par le terminal. Pour se protéger contre cette menace, il faut employer des protocoles d'authentification plus élaborés que la présentation d'un simple mot de passe (par exemple des protocoles basés sur la carte à microprocesseur : voir section suivante, page 85).

Les mots de passe doivent être fréquemment changés pour ne pas permettre à un fraudeur de s'introduire durant une longue période dans le système. La meilleure solution serait de modifier le mot de passe à chaque connexion. Cette méthode, qui semble compliquée à implanter, a trouvé une solution pratique en utilisant des dispositifs matériels.

---

8) Les attaques contre les mots de passe sont décrites au point 2.3 du chapitre 2 (première partie), ainsi que quelques solutions bien connues permettant de les contrer.

5.1.1.2. Le contrôle matériel

Dans le cadre de ce contrôle, le système de protection cherche à vérifier qu'un utilisateur est bien en possession d'un élément matériel qui puisse garantir son identité.

Les systèmes qui existent actuellement s'apparentent au matériel baptisé GRENOUILLE de la société MUSTANG TECHNOLOGIE et dont nous allons présenter le fonctionnement. Des systèmes plus anciens utilisent le principe de la clé et du cadenas; on verrouille l'accès au système, seules les personnes possédant la clé (une carte) peuvent y accéder.

5.1.1.2.1. Principe

La Grenouille est un appareil de petites dimensions (quatre centimètres de diamètre, moins d'un centimètre d'épaisseur). Sa face avant présente un affichage à cristaux liquides (LCD) alors que sur sa face arrière on trouve des cellules photosensibles. L'ensemble contient une unité arithmétique et logique alimentée par une petite pile au mercure.

Le système se compose en plus d'une partie logicielle qui sera implantée dans la procédure classique de "LOGIN". Le partie logicielle va émettre vers le terminal un signal optique correspondant à un nombre aléatoire. Le processus de contrôle se déroule de la manière suivante :

a) L'utilisateur demande une connexion à partir de son terminal;
b) un message lui est affiché, ainsi qu'un mire lui indiquant l'endroit de l'écran sur lequel il doit placer sa Grenouille;
c) la Grenouille signale à l'utilisateur qu'elle est bien placée; celui-ci tape alors un retour chariot;
d) le programme de "LOGIN" émet un signal optique via le terminal vers la Grenouille qui affiche immédiatement le résultat d'un calcul interne.

C'est ce résultat affiché qui sera utilisé comme mot de passe par l'utilisateur. Du fait du caractère aléatoire du signal optique généré, le code résultant du calcul de la Grenouille et introduit au clavier est différent à chaque nouvelle utilisation; il n'est donc pas nécessaire de chiffrer les messages qui transitent sur la liaison.

5.1.1.2.2. Limites de la méthode

Ces techniques sont relativement récentes. Les systèmes externes tels que la Grenouille présentent des avantages incomparables par rapport à d'autres techniques plus anciennes, telles que les badges magnétiques; il n'est plus nécessaire d'utiliser de lecteur, ce qui diminue le prix de revient. Contrairement aux cartes magnétiques, ils ne comportent pas d'informations que l'on puisse lire et reproduire. Le dialogue avec la routine sécurisée est différent pour chaque demande d'identification.

La sécurité de ces systèmes tient à la manière de noyer le code de vérification. Les systèmes multitâches de grandes dimensions peuvent faire exécuter les routines de vérification en mode superviseur et les ajouter au système de base afin de contrôler périodiquement les utilisateurs connectés.

Le nombre de combinaisons optiques que peut générer un écran traditionnel est relativement limité, et le fraudeur peut faire des recherches exhaustives, en émettant vers son écran différents signaux et en mémorisant les résultats calculés par la Grenouille.

Ces dispositifs ne peuvent pas être invalidés, ce qui est une faille supplémentaire, car tout le système est détruit en cas de perte ou de vol d'un seul d'entre eux.

5.1.1.3. Les procédés biométriques

Depuis l'apparition de la police scientifique, on sait que l'on peut identifier un individu par la reconnaissance de certaines caractéristiques physiques. Ainsi, les empreintes digitales ont été longtemps un moyen efficace d'identification car il était relativement facile à mettre en oeuvre.

5.1.1.3.1. Principe de fonctionnement

L'évolution de la technique a permis de développer des appareils capables de reconnaître des caractéristiques dont l'acquisition est plus complexe que celle qui consiste à passer son doigt sur un tampon encreur pour le déposer ensuite sur un formulaire.

Parmi ces techniques, on peut citer la reconnaissance de la voix, l'analyse vasculaire de l'oeil ou la reconnaissance automatique de signatures

[ACH-86]. Ces techniques n'ont pu se développer que de manière expérimentale car le coût des appareils capables de faire la saisie est encore prohibitif.

Pour mettre en place de telles méthodes, il faut préenregistrer les caractéristiques physiques sur un support informatique en correspondance avec l'identificateur de la personne. Lorsqu'elle se présente, on lui demande d'entrer cet identificateur (par exemple par l'intermédiaire d'un clavier). Cela permet de retrouver les caractéristiques enregistrées. On effectue alors la mesure réelle que l'on compare à celle qu'on vient d'extraire du fichier.

Cette façon de procéder réduit le temps de recherche de l'information. De plus, puisque le système connaît a priori l'ensemble des caractéristiques, il peut se contenter de n'en tester que quelques-unes et de ne mesurer que des échantillons. La taille de ces échantillons est bien sûr liée à la nature des caractéristiques physiques testées. La probabilité de se tromper doit rester très faible en regard du coût total du système.

5.1.1.3.2. Limites de la méthode

Les procédés biométriques sont chers à réaliser si l'on veut obtenir une bonne sécurité. Par exemple, dans le cas de la reconnaissance de signatures manuscrites, on ne peut pas se contenter de reconnaître le déplacement d'un stylo sur une tablette. Il faut également mesurer la pression exercée en différents points du paraphe, les accélérations, l'inclinaison, la vitesse, les changements de direction du stylo, etc. [ACH-86].

Ce sont les seules techniques qui permettent une réelle identification alors que les autres ne peuvent qu'authentifier le possesseur d'un droit. La parade à ces mesures relève de la science-fiction, tant il est aujourd'hui difficile de recréer parfaitement les caractéristiques physiques d'un individu. On réservera ces méthodes à des applications qui nécessitent un très haut degré de sécurité, pour lesquelles le coût n'est pas un paramètre essentiel.

5.1.2. La protection inter-utilisateurs du système d'exploitation

Dans un contexte de partage du système entre plusieurs utilisateurs, il est indispensable que chaque usager puisse être protégé des autres en conservant la "propriété" des ressources qui lui ont été attribuées. La

plupart des systèmes d'exploitation fournissent donc des mécanismes de vérification des droits d'accès de tout utilisateur aux ressources qu'il désire utiliser.

5.1.2.1. Schéma général

Le schéma classique de ce système de protection est présenté à la figure 5.9.; c'est celui du *moniteur de référence* ("Reference monitor") [VAX-85] qui permet de décrire le système en termes de *sujets* et d'*objets* (9).

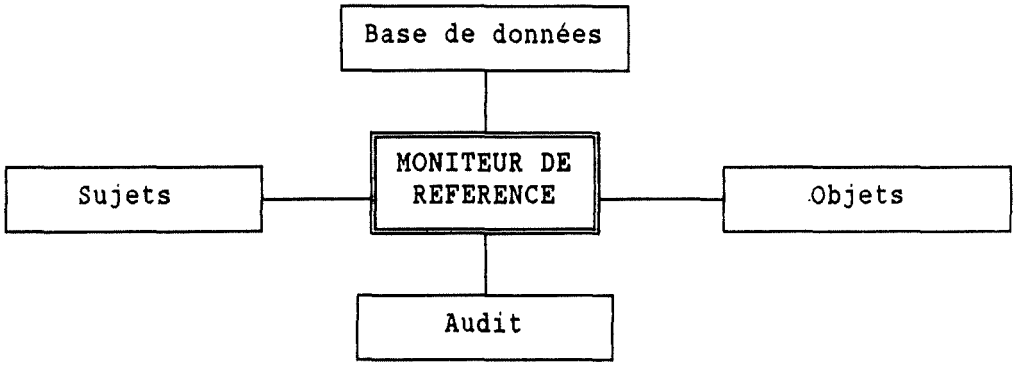

Figure 5.9. : Le Moniteur de Référence

Une base de données détermine les droits d'accès que possèdent les sujets sur les objets. Une fonction d'audit permettant de surveiller les accès (ayant ou n'ayant pas abouti) qui font appel à la base de données des droits d'accès.

9) On entend ici par *objet* des ressources matérielles (disques, cassettes, processeur, ...) ou des ressouces abstraites (structures de données, traitement des données, sémaphores, fichiers, bibliothèques, ...). De même, le mot *sujet* désignera ici une entité active (ressource ou autre) qui accède à l'information au nom de l'utilisateur (utilisateur, programme, processus ...).

5.1.2.2. Principe de fonctionnement

Le scénario de fonctionnement typique de ce mécanisme est présenté à la figure 5.10. [IBM-83]

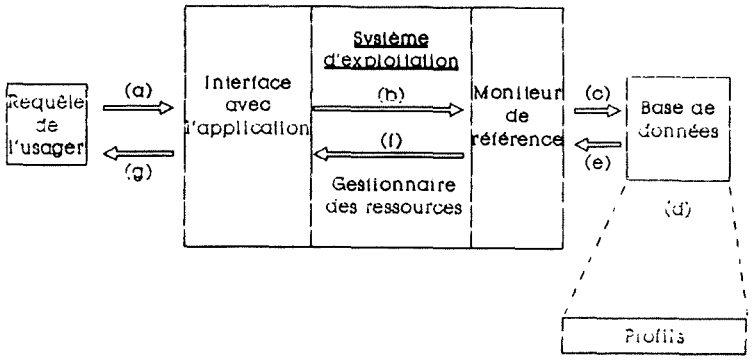

Figure 5.10. : Fonctionnement général du système

Un sujet demande l'accès à un objet (a), au système d'exploitation, qui passe la requête au moniteur de référence (b). Celui-ci se réfère à sa base de données (c) et vérifie les droits qu'a l'utilisateur d'accéder à l'objet désiré (le *profil* de l'utilisateur) (d). En se basant sur les renseignements de ce profil (e), le moniteur de référence passe le *statut* de la requête au système d'exploitation (l'utilisateur peut ou ne peut pas accéder à l'objet) (f); le système d'exploitation *sert* cette requête, ou la refuse le cas échéant (g).

5.1.2.3. Contrôle d'accès aux logiciels

Ce mécanisme de protection peut être utilisé directement pour contrôler l'accès au logiciel que l'on désire protéger. Si l'on veut par exemple autoriser l'accès au logiciel L par les utilisateurs A, B et C mais le refuser aux utilisateurs D et E, il suffit d'en informer le moniteur de référence. Celui-ci conservera cette information dans la base de données des droits d'accès; il saura ainsi que le statut d'une demande d'accès à L émise par B est "accès autorisé", alors que si elle avait émané de D le diagnostic aurait été "accès refusé".

5.2. Protection par le logiciel lui-même

5.2.1. Principe

Une autre manière de protéger un logiciel contre son utilisation par des personnes non autorisées est d'implanter un mécanisme de contrôle d'accès directement dans le logiciel. Celui-ci refusera alors de s'exécuter si la procédure d'identification n'est pas satisfaite.

Cette solution est plutôt utilisée par les distributeurs de logiciels pour limiter le piratage. Elle peut également être appliquée dans un certain nombre de cas où la sécurité doit être maximale, la vérification "normale" du système d'exploitation étant alors doublée d'un contrôle d'accès spécifique pour les logiciels "sensibles". Un environnement où aucune protection n'est offerte par le système d'exploitation peut de même être sécurisé par l'implantation de protections d'accès spécifiques à certains logiciels (par exemple, un contexte micro-ordinateur).

5.2.2. Limites de la méthode

Le principale difficulté rencontrée par le concepteur d'une telle méthode est de noyer efficacement la routine de vérification au sein du code du logiciel. En effet, une personne suffisamment patiente pourra analyser le logiciel instruction par instruction, localiser la fonction de vérification et la supprimer. Le concepteur doit rendre ce travail de *déplombage* suffisamment difficile (et donc coûteux) pour annuler le bénéfice à retirer de cette opération.

5.3. Conclusion

L'accès à un logiciel peut être protégé par le logiciel lui-même ou par l'utilisation de mécanismes du système d'exploitation. Ces derniers offrent une sécurité acceptable dans bon nombre de cas, pour un coût inférieur.

Il convient de plus de se souvenir que tout type de protection d'accès est subordonnée à une identification préalable, qui conditionne la sécurité

de tout le système. La qualité du système de protection d'accès ne pourrait être meilleure que celle du système d'identification sur lequel il se base.

6. La carte à mémoire

La protection des logiciels fait intervenir, nous l'avons vu, différents problèmes relatifs à la génération, à la confidentialité, au transport et à la distribution de clés cryptographiques, à l'identification des personnes, à la vérification de droits d'accès, etc. La technologie des cartes à mémoire peut y apporter une solution, c'est pourquoi nous la présentons brièvement ici en guise d'illustration.

6.1. Généralités

Le terme *carte à mémoire* regroupe un ensemble d'objets qui se présentent sous la forme d'un rectangle de plastique au format des cartes de crédit traditionnelles, et dont l'interface externe est constituée d'un ensemble de huit contacts disposés dans le coin supérieur gauche. Leur originalité est de substituer à la piste magnétique classique un micro-circuit électronique capable de mémoriser des informations.

Deux technologies de cartes à mémoire prédominent actuellement : les cartes à logique câblée, et les cartes à microprocesseur.

6.2. Les cartes à logique câblée [LOU-88]

Elles sont ou non protégées vis-à-vis des accès. Les enchaînements des différentes étapes des procédures d'accès y sont figés dans des circuits électroniques. L'ensemble mémoire plus procédure d'accès est donc fabriqué dans le seul but de stocker et de protéger des données. Le milieu extérieur dialogue avec les circuits électroniques, distincts de la partie mémoire et qui en contrôlent l'accès.

Ce type de carte est bien adapté pour le stockage et la protection des données. Il présente l'inconvénient de ne pas être universel. En effet, à chaque changement d'application, on est obligé de réétudier la logique d'accès et d'effectuer une nouvelle intégration des composants qui l'exécutent. Par contre, sa structure simplifiée doit permettre sa distribution à des coûts compétitifs. La TELECARTE commercialisée par FRANCE-TELECOM est un

exemple concret d'utilisation à grande échelle de cette technique.

6.3. Les cartes à microprocesseur

Les cartes à microprocesseur sont des cartes dans lesquelles on a inséré un composant comportant les éléments constitutifs d'un véritable ordinateur. En plus de la mission de mémorisation, l'électronique sera donc capable d'exécuter des programmes simples. La mémoire est protégée de l'extérieur par le programme exécuté par l'unité de traitement. On peut alors inscrire des paramètres de fonctionnement secrets dans cette mémoire en sachant que leur intégrité et leur confidentialité sont assurées par le programme interne [BAU-87].

La carte à microprocesseur est donc bien adaptée aux applications qui requièrent un haut degré de protection. Les fonctionnalités de cette carte, qui sont liées au programme enregistré lors de sa fabrication, permettent d'imaginer de nombreuses utilisations.

6.4. La carte à microprocesseur pour la sécurité des logiciels

On retrouve la carte à microprocesseur dans toutes les applications qui requièrent un haut degré de sécurité [LOU-88]. D'une manière générale, la carte est capable de calculer une fonction sur des éléments secrets et d'autres connus. Cette fonction a la forme :

R = F(données internes,données externes,secret).

Si cette fonction F est non inversible, elle peut être utilisée pour réaliser l'authentification ou la certification de la carte, ou encore pour calculer des signatures électroniques.

6.4.1. L'authentification

L'authentification d'une carte est une procédure destinée à vérifier que la carte est détentrice d'un secret. Dans l'affirmative, la procédure de vérification en déduira que le possesseur de la carte est bien ce qu'il

prétend être.

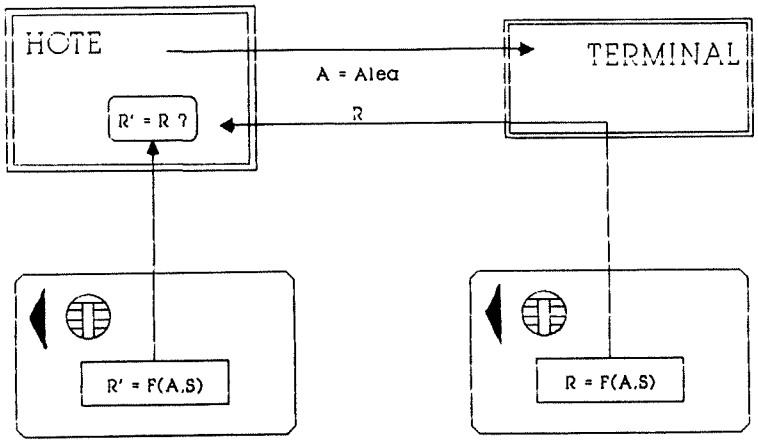

Figure 5.11. : Procédure d'authentification

Dans ce protocole, l'hôte veut authentifier la carte de l'utilisateur présentée via le terminal. Il commence par tirer un nombre aléatoire A, qu'il transmet au terminal et à un module de sécurité interne. Les deux cartes effectuent un calcul sur ce nombre et sur un secret S, enregistré dans leur mémoire. La carte du côté terminal envoie le résultat R de son calcul que l'hôte compare au résultat R' de son module de sécurité. Si R = R', l'hôte déduira que la carte possède le même secret que lui, qu'elle est donc authentique et que son porteur est autorisé à accéder à la ressource demandée.

6.4.2. La certification

La certification permet de savoir si la carte présentée possède bien un mot particulier inscrit dans sa mémoire. Ce mot peut être un droit

d'utilisation d'un logiciel, une somme (pour une carte bancaire), etc.

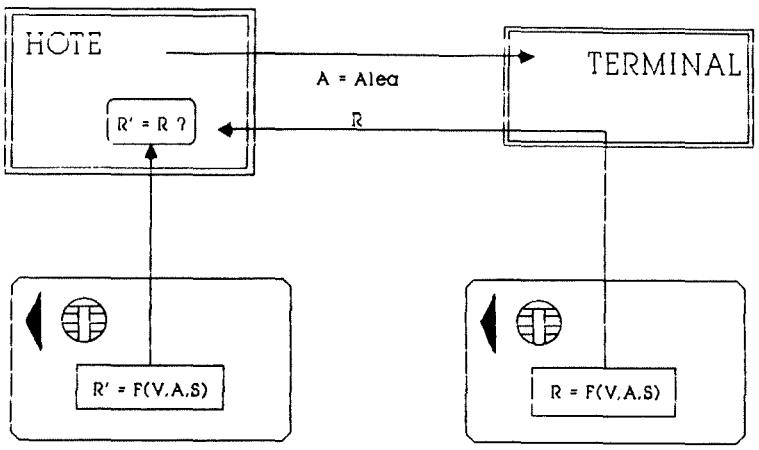

Figure 5.12. : Procédure de certification

L'hôte désire s'assurer que la carte présentée a bien dans sa mémoire la valeur V, qu'elle prétend détenir. Il commence par demander une lecture du mot puis émet vers la carte un nombre aléatoire A. La carte calcule le résultat en utilisant son secret S, A et la valeur V qui se trouve à l'adresse mémoire prévue. Le terminal transmet le résultat à l'hôte qui le compare avec celui qu'il a fait calculer à la carte de référence par l'intermédiaire d'un module de sécurité.

6.4.3. La signature électronique

La signature électronique, comme la signature manuscrite, permet de s'assurer que le message transmis a bien été expédié par la bonne personne; le cas échéant, elle constituera une preuve devant un tribunal, à la manière d'une signature manuelle classique. Cette fonction de la carte peut être utilisée dans le dialogue hôte-terminal, par exemple au cours de la procédure

d'authentification, pour éviter l'injection de faux messages sur la ligne.

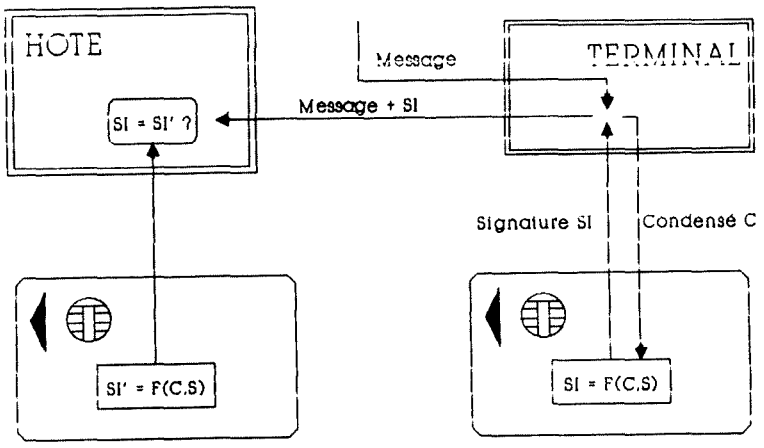

Figure 5.13. : Procédure de signature

Le terminal envisagé dispose dans cet exemple de moyens de calcul. Il envoie à la carte un condensé du message, obtenu par une fonction de condensation (10). La carte calcule une signature SI à partir du condensé C et de son secret S. Le message et sa signature sont transmis à l'hôte qui, grâce à une carte de référence possédant le même secret, peut comparer la signature reçue à celle que son module de sécurité a calculée (11).

6.4.4. Le chiffrement

Ce dernier exemple n'est envisageable que si l'on dispose d'une fonction réversible.

Le chiffrement est le moyen le plus sûr de sécuriser les informations qui transitent sur un réseau. On utilise aussi cette technique pour rendre inaccessibles les informations de certains fichiers ou logiciels sensibles

---

10) Voir la définition de fonction de condensation au point 4.2.2.2. du présent chapitre.

11) L'utilisation d'un cryptosystème à clé publique se prête mieux à la réalisation d'un schéma de signature (par exemple, RSA) [POZ-87, POZ-86]. Malheureusement, des limitations techniques (performances et capacité de stockage trop limitées de la carte) rendent son utilisation par la carte impossible à l'heure actuelle.

(ou une partie critique de ceux-ci).

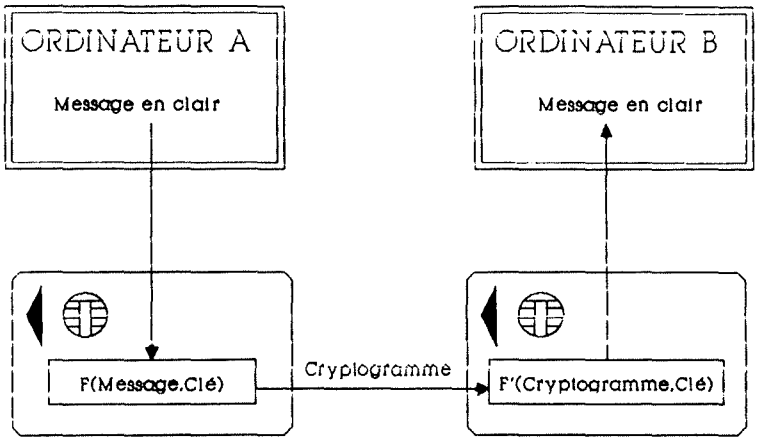

Figure 5.14. : Procédure de chiffrement

Le message en clair est découpé en blocs qui sont chiffrés par la carte à partir de la fonction F en utilisant une clé secrète. Le message chiffré est transmis à l'autre bout de la ligne où une carte possédant une clé de déchiffrement exécute le calcul inverse.

Remarquons de nouveau ici que les faibles performances des cartes actuelles rendent cette procédure très lourde. La plupart des applications pratiques devront adapter le principe en faisant calculer les fonctions F et F' par un module extérieur à la carte, celle-ci se contentant de jouer le rôle de "porte-clés" de haute sécurité [FER-86].

6.5. Les avantages de la carte à microprocesseur

La carte à microprocesseur présente des avantages d'utilisation par rapport à tous les dispositifs qui servent à la protection d'accès.

6.5.1. La facilité de transport

La carte est du format d'une carte de crédit, ce qui la rend facilement transportable en toutes circonstances. Elle est très peu sensible aux perturbations magnétiques; il n'est donc pas nécessaire de prendre des précautions pour la ranger.

Les dispositifs externes tels que les bouchons ou les systèmes comme la Grenouille ne disposent pas de cette facilité.

6.5.2. L'intégrité et la confidentialité des données

La carte dispose de moyens perfectionnés qui protègent l'intégrité des données qu'elle mémorise. C'est un support sûr dans la mesure où l'on ne peut plus modifier les valeurs qui y ont été enregistrées. Les mécanismes de protection sécurisent également l'écriture de nouvelles informations dans la mémoire. Le composant a de plus été conçu en superposant les différentes mémoires entre elles et avec les autres circuits; l'espionnage avec un microscope électronique est à cause de cela difficilement réalisable.

L'emploi de la carte est donc recommandé pour transporter des informations secrètes ou dont l'intégrité doit être protégée, telles des droits d'utilisation, des clés cryptographiques, ou encore les résultats issus du contrôle d'intégrité des logiciels ("*checksums*").

6.5.3. La non reproductibilité de la carte

La technologie de la carte à microprocesseur est complexe à maîtriser; il est donc très difficile pour des fraudeurs d'arriver à en fabriquer de nouvelles. De plus, les fabricants de cartes se sont dotés d'installations perfectionnées pour assurer un très haut niveau de sécurité, tant au niveau de la fabrication que du stockage et du transport [LOU-88].

La carte est pour l'instant un support unique du point de vue de la sécurité puisqu'il n'est pas possible de simuler son comportement avec des composants traditionnels en respectant le format. Le fraudeur ne pourrait pas introduire la copie dans des lecteurs implantés en milieu public.

Cet avantage est important par rapport à des dispositifs "bouchon" (par exemple, le DONGLE programmable d'ELECTRYON, qui contient une mémoire morte avec des clés).

6.5.4. L'unicité du dialogue

Le dialogue qui s'établit avec la carte peut être espionné. De plus, les protocoles sont normalisés et les différentes commandes sont documentées. Ce dialogue n'est cependant pas reproductible dans son ensemble, car les procédés d'authentification font intervenir des variables aléatoires qui modifient à chaque fois les réponses de la carte.

La carte présente donc un intérêt lors des procédures d'authentification des utilisateurs puisqu'elle supprime la possibilité de *rejeu*.

6.5.5. Une large étendue de possibilités

La carte dispose de deux atouts : sa capacité de mémorisation et sa fonction de calcul.

Ces deux fonctions peuvent être utilisées conjointement pour réaliser des applications sophistiquées. En combinant divers secrets et en ajoutant des variables dans les zones de la mémoire, on peut définir et organiser une hiérarchie de droits d'utilisation pour des machines ou pour des logiciels. Ces droits pourront avoir des périodes de validité si on leur associe des dates de péremption.

A la manière de la TELECARTE de FRANCE TELECOM, il est également possible de réaliser des applications qui utilisent une sorte de crédit logiciel.

6.5.6. La facilité d'utilisation

L'investissement de départ consiste en l'achat d'un lecteur pour chaque terminal. Notons que ces lecteurs existent à présent sous forme intégrée dans des micro-ordinateurs, à la manière des lecteurs de disquettes.

Les cartes, lorsqu'elles sont commandées par centaines, reviennent à moins de 100 FF l'unité (environ 600 FB). Pour une application donnée, on peut rapidement créer de nouvelles cartes et donc introduire de nouveaux utilisateurs dans le système. Il faut cependant avoir recours à une structure, avec une personne responsable de l'attribution des cartes et de leur personnalisation.

6.6. Evolution de la technologie

Au début de l'utilisation de cette technique, on pressentait que l'évolution de la carte à microprocesseur irait vers une augmentation de sa capacité de stockage. En fait, il semble que l'on va plutôt vers une augmentation de ses possibilités de calcul. En particulier, on commence à introduire dans les cartes des algorithmes de chiffrement tels le DES ou le RSA [FER-86].

Il semble cependant que les performances de ce type de composant resteront une caractéristique bloquante pour certaines applications. Ceci est principalement dû au fait que la carte actuelle communique avec le milieu extérieur via une connexion physique assurant à la fois l'entrée et la sortie des ordres et des données sur un seul fil. On a vu des annonces de fabricants japonais à propos de cartes à microprocesseur dialoguant avec l'extérieur par des liaisons parallèles; ces cartes pourraient donc être beaucoup plus rapides. Toutefois, la fabrication et l'utilisation de ce type de composant se heurte aux normes internationales en la matière. Il semble donc peu probable que ces cartes effectuent une rapide percée sur le marché.

6.7. Conclusion

Les deux caractéristiques de la carte à microprocesseur (stockage sûr et moyens de calcul) en font un "must" pour les applications nécessitant un degré de sécurité élevé. Son faible coût associé à une nette suprématie sur les méthodes plus conventionnelles laissent entrevoir un essor important de cette technologie dans les années à venir.

7. Conclusion

Après avoir présenté quelques notions de base en cryptographie, nous avons proposé quatre axes de protection techniques des logiciels contre la malveillance.

Ainsi, la confidentialité d'un logiciel peut-elle être protégée par chiffrement ou selon d'autres techniques, comme la fragmentation-dissémination. Un cas particulier de protection de la confidentialité garantit le droit du distributeur légitime du logiciel.

L'intégrité est certifiée lorsque le logiciel est rendu immuable grâce à certaines caractéristiques de son support. Un contrôle de l'intégrité peut aussi être réalisé au moyen de fonctions cryptographiques. De plus, l'accès au logiciel peut être protégé, dans le but d'interdire son exécution aux personnes non autorisées. Le système d'exploitation ou le logiciel lui-même peuvent effectuer ce contrôle.

La plupart de ces méthodes de protection peuvent être réalisées de manière plus sûre grâce aux caractéristiques techniques de la carte à microprocesseur.

CHAPITRE III

Protections contre les virus informatiques

CHAPITRE III : LA PROTECTION CONTRE LES VIRUS INFORMATIQUES

Nous avons vu que les virus constituent un grave problème pour la sécurité des systèmes informatiques en général, et des logiciels en particulier. Cette section a pour but d'exposer brièvement quelques moyens de protection contre ces programmes infectieux.

Nous étudierons tout d'abord les techniques de protection universelles. Nous tenterons ensuite d'élargir les hypothèses restrictives qui auront dû être posées; ceci nous conduira à exposer quelques méthodes plus souples, mais incapables d'offrir une protection efficace dans tous les cas. Nous terminerons en proposant quelques recommandations aux personnes désireuses de protéger un système contre la menace des virus.

1. Les méthodes universelles

Nous allons examiner dans cette section les méthodes permettant de protéger contre n'importe quel virus tout système dans lequel elles seraient appliquées à la lettre. Ces méthodes sont à classer en deux groupes; les premières agissent a priori contre l'infection proprement dite, les secondes agissent a posteriori par la détection de toute modification.

1.1. Méthodes de protection a priori

Nous avons dit plus haut (première partie, chapitre 3 (page 38)) qu'une infection pouvait avoir lieu dès qu'un partage d'information pouvant être interprétée de manière générale était autorisé.

Cette affirmation nous donne quelques moyens d'action efficaces pour la prévention d'une attaque virale. Nous verrons malheureusement que ces méthodes introduisent de telles limitations dans l'utilisation du système qu'elles ne seront applicables que dans un nombre très restreint de cas.

1.1.1. L'isolationnisme

Remarquons tout d'abord que s'il n'y a pas de partage autorisé alors aucune information externe ne peut être interprétée. Dans ce cas une contamination venue de l'extérieur du système est donc impossible.
L'application d'une telle technique de protection, qualifiée d'isolationniste, est totalement inacceptable dans une immense majorité de cas car elle interdit aux utilisateurs de bénéficier du travail d'autres personnes.

1.1.2. Non généralité d'interprétation de l'information

Une autre technique universelle de protection est de restreindre la possibilité d'interpréter librement l'information, puisque cette interprétation est nécessaire à l'infection (par exemple, le virus est tantôt interprété comme une donnée, au moment où il est recopié, tantôt comme un programme lorsqu'il s'exécute. Il suffirait par exemple d'interdire son interprétation comme une donnée pour rendre l'infection impossible). Mais l'hypothèse de généralité est indispensable dans une machine de Turing [TUR-50]; utiliser cette technique de protection nous conduit donc à interdire aux ordinateurs d'agir comme des machines de Turing. Concrètement, cela veut dire qu'il serait par exemple impossible d'interpréter du code exécutable comme "donnée", ce qui aurait pour conséquence de rendre impossible la construction d'un *loader*. Il semble donc que cette solution soit à rejeter dans un système informatique digne de ce nom.

1.1.3. Non modifiabilité des exécutables

Une variante des deux techniques présentées ci-dessus serait de définir quels sont les objets (1) susceptibles d'être interprétés comme programme, et d'empêcher toute modification de ceux-ci. Cette solution est proposée par POZZO et GRAY [POZ-86, POZ-87]; elle découle du fait que l'infection est réalisée en modifiant de l'information qui sera par la suite interprétée

---

1) Le mot "objet" fera ici aussi référence à une entité cohérente d'information (comme un fichier ou un programme, par exemple).

comme un programme (2).

Cette solution n'est pas praticable en tant que moyen de protection universel car elle introduit des limitations trop importantes sur l'utilisation du système. Nous en reparlerons donc dans la section traitant des moyens de protection non universels (page 104).

1.2. Méthodes de protection a posteriori

Nous ne disposons donc, dans la majorité des cas, d'aucun moyen efficace acceptable pour protéger un système contre toute contamination externe quelle qu'elle soit. L'idée est à présent de se demander s'il n'existe pas des méthodes permettant de détecter une contamination éventuelle. cette détection serait utilisée en conjugaison avec des techniques dont le but est de restaurer le système dans un état sain avant que la contagion n'aie lieu.

1.2.1. Le chiffrement

POZZO et GRAY proposent l'utilisation de méthodes cryptographiques pour détecter toute modification d'un exécutable avant son utilisation. Notons une fois encore que nous prenons ici la notion "d'exécutable" au sens large de "tout objet susceptible d'être interprété comme un programme". Or, dans un contexte où la généralité de l'interprétation est de mise, cette propriété s'étend à tout objet.

Le principe de la méthode est décrit à la figure 6.1. Un objet f est chiffré par un système cryptographique; nous appelons $E_K(f)$ le résultat de ce chiffrement. Lorsque l'on désire interpréter cet objet f ("l'exécuter"), c'est $E_K(f)$ qui est transmis à l'environnement d'exécution; celui-ci émet une requête de déchiffrement au système cryptographique, puis tente d'exécuter le résultat.

Un virus ayant infecté $E_K(f)$ est rendu sans signification par le déchiffrement. Une erreur sera alors très probablement détectée par l'environnement d'exécution lorsqu'il tentera d'utiliser la partie modifiée de

---

2) POZZO et GRAY parlent seulement d'*exécutables*. Mais une remarque de COHEN qui dit que *protéger seulement les exécutables, les programmes source et les fichiers intermédiaires ne préviendra pas les virus* [COH-88], nous a conduit à élargir leur propos.

l'exécutable, et l'autorité responsable pourra être notifiée afin qu'elle prenne les mesures de désinfection qui s'imposent.

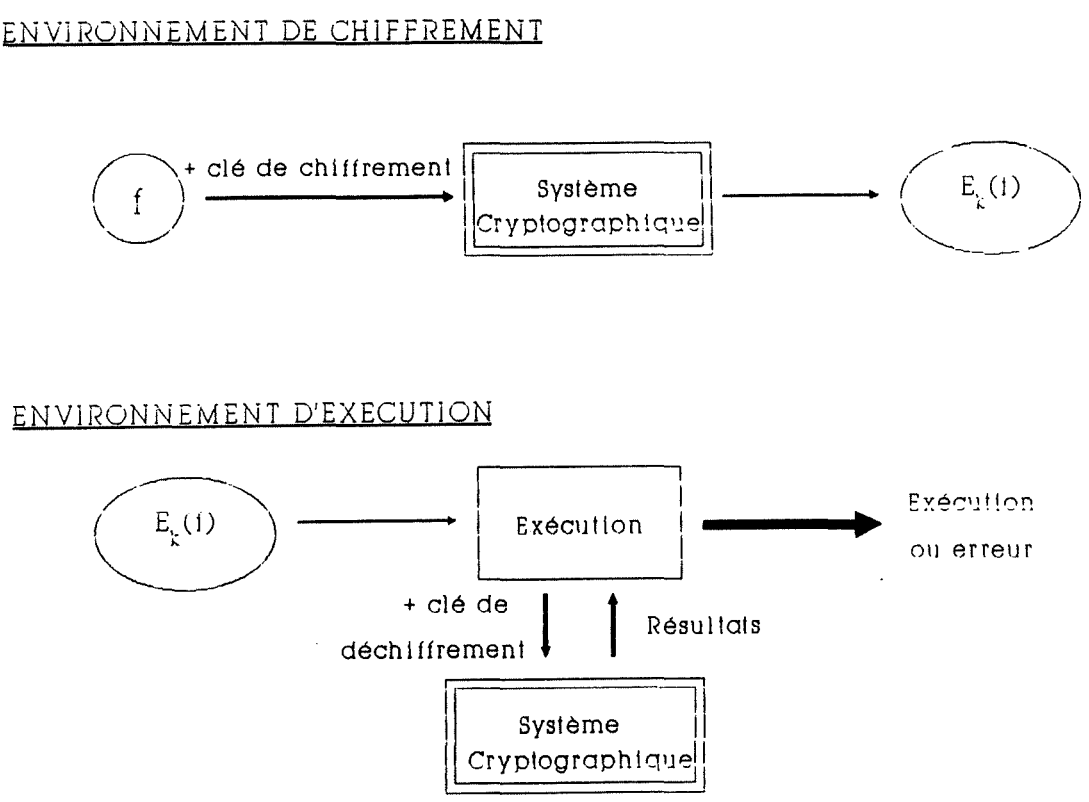

Figure 6.1. : Mécanisme de détection

L'avantage principal de cette méthode est qu'elle relie le moyen de protection à l'objet à protéger plutôt qu'au système ou au moyen de stockage; son utilisabilité dépend donc moins de la sécurité du système qui la supporte. De plus, les modifications des exécutables peuvent toujours être détectées en-dehors du domaine d'un système particulier, par exemple au site destination lors d'un transfert d'information. Cette méthode offre en outre l'avantage de détecter toute modification des objets protégés, même non imputables à un virus.

Le défaut inhérent à ce type de protection a déjà été mentionné; il s'agit du fait qu'aucune modification n'est empêchée, et donc qu'un refus de service aux utilisateurs a lieu lors d'une tentative d'infection. Il faut alors mettre en oeuvre une politique permettant de restaurer la système dans son état initial.

Le moyen le plus simple de réaliser une telle politique est de conser-

ver une copie de sauvegarde saine de chaque objet, dans un endroit offrant une totale garantie d'intégrité. On peut par exemple juger qu'une armoire répond à cette exigence (on ne considère pas ici la possibilité d'une malveillance de la part d'une personne ayant accès au système car il n'est alors pas nécessaire pour celle-ci de construire un virus pour lui nuire [FAK-88]).

Remarquons ici que le schéma de protection tel qu'il est décrit n'est efficace que dans la mesure où aucun fichier chiffré ne contient de virus. Dans ce cas toute infection est en effet stoppée, comme expliqué à la figure 6.2.

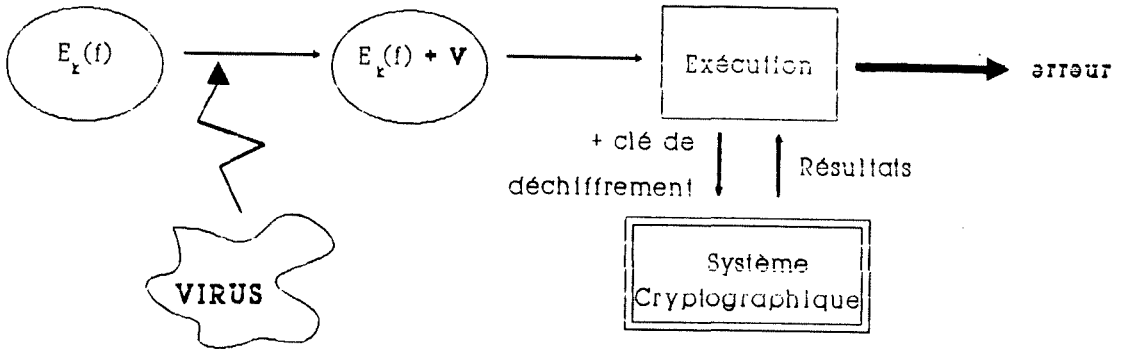

Figure 6.2. : Tentative d'infection d'un fichier sain

Par contre, lorsqu'un objet déjà infecté est chiffré, une interprétation de cet objet comme un programme causera l'exécution du virus sans que

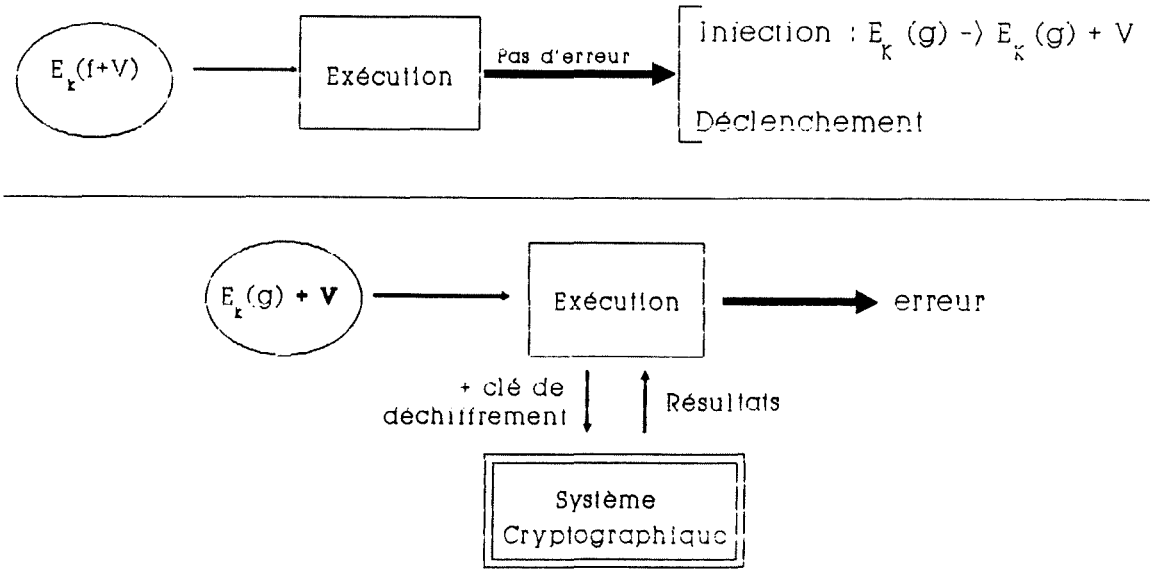

Figure 6.3. : L'infection lorsqu'un virus est chiffré

celui-ci ne soit détecté, et par là une tentative d'infection (voir figure 6.3.). Cette exécution du virus aura également pour effet de permettre le déclenchement prévu par le concepteur, ce qui est plus grave, car la tentative d'infection sera détectée lors de la vérification de sa cible, lorsque celle-ci sera utilisée; mais le déclenchement du virus pourra, lui, avoir lieu sans que rien ne puisse l'empêcher. Cette technique de protection des exécutables n'est donc efficace que si elle est appliquée à un système sain dès le départ. Elle permet alors une détection des modifications subies par un fichier avant son interprétation.

1.2.2. La signature

Une réalisation possible de cette technique de détection des modifications serait de réaliser une signature des fichiers à protéger ([3]). POZZO et GRAY conseillent d'utiliser pour cela le schéma suivant :

> On applique une robuste fonction cryptographique à sens unique à l'objet à protéger concaténé à des informations additionnelles comme l'identité du signataire et une date. La "checksum" ainsi obtenue est alors chiffrée selon un algorithme à clé publique, puis cette signature est accolée à l'objet protégé.
>
> Au moment de l'exécution, la signature est déchiffrée au moyen de la clé publique, la checksum est recalculée et les deux valeurs comparées. Si celles-ci diffèrent, c'est que l'objet a été modifié; l'autorité responsable sera alors avertie et l'exécution n'aura pas lieu.

Cette méthode offre l'avantage d'un large degré de flexibilité dans la détermination de la liste des clés publiques. De plus, la confidentialité de cette liste ne doit pas être protégée par le système; seule son intégrité doit être prise en charge.

Comparée à un chiffrement pur et simple des objets, cette implantation offre comme avantage que l'environnement d'exécution ne tentera jamais d'utiliser de l'information "aléatoire" résultant du déchiffrement d'information structurée non chiffrée (le code du virus) car la détection a ici

---

[3]) Un exemple concret simplifié est proposé par COHEN. Il s'agit d'un module écrit en C qui permet de réaliser une vérification automatique de tout programme dans lequel il est inclus, en testant la validité d'une signature RSA stockée par le système [COH-88].

lieu avant toute interprétation de l'objet.

1.2.3. Evaluation de ces techniques

L'application de ces méthodes de protection par la cryptographie pose un certain nombre de problèmes.

Nous avons mentionné ci-dessus la nécessité de restaurer un système dans lequel le mécanisme de protection a détecté une modification. Une manière de procéder lorsque cette modification est due à un virus est de tenter de désinfecter le fichier modifié. En effet, il existe un moyen de désinfection quel que soit le virus à éliminer [COH-87]. Malheureusement, nous verrons que la détection automatique des virus est indécidable; ce résultat peut d'ores et déjà être prolongé, en disant qu'étant donné un programme que l'on sait infecté par un virus inconnu, il est impossible de déterminer automatiquement et pour tout programme à quel endroit de celui-ci se trouve le virus. Par conséquent, il est impossible de créer un programme de désinfection qui serait efficace contre tous les virus qu'il est possible de créer. Cette solution souple est donc inapplicable si l'on désire une protection totale.

Une autre manière de procéder à une restauration du système dans son état initial consiste à remplacer les fichiers infectés par des copies de sauvegarde réalisées dans l'état initial de ce système. Cette méthode est simple à mettre en oeuvre, mais elle est parfois lourde car il est indispensable de disposer d'une copie de chaque objet, suffisamment récente pour neutraliser les dégâts causés par l'infection. La dernière version saine de chaque objet doit donc avoir été sauvegardée, ce qui n'est pas sans poser des problèmes de performances dans un environnement où beaucoup de modifications ont lieu. De plus, cette technique nécessite l'intervention manuelle d'un opérateur car les copies de sauvegarde doivent impérativement se trouver physiquement hors d'accès de chaque programme afin d'être à l'abri de toute infection virale.

Cette technique assez lourde peut être assouplie sans perte de sécurité si l'on accepte l'éventualité de perdre quelques versions d'un objet quelconque. Ainsi, dans un système au sein duquel les fichiers sont sujets à beaucoup de modifications, les copies de sauvegarde ne seront-elle réalisées qu'à intervalles réguliers (par exemple tous les soirs). On acceptera ainsi de perdre le travail d'une journée dans le cas d'une attaque virale, avec

pour bénéfice un gain appréciable de temps lorsque tout se passe bien. Cette technique offre l'avantage de ne pas changer la manière habituelle de travailler; une politique de sauvegarde régulière est en effet utilisée dans dans tous les systèmes informatiques car elle permet la reconstruction d'un système en cas d'incident ("crash" des disques, par exemple).

Le deuxième problème posé par l'application de la protection par la cryptographie est plutôt de type organisationnel ou du moins relève de la gestion du système. Il est en effet indispensable de n'accorder le droit de protéger des fichiers qu'à des utilisateurs "sûrs" afin de limiter le risque qu'un objet infecté ne puisse être légitimement chiffré. Nous avons en effet déjà montré que le déclenchement d'un virus chiffré n'est pas bloqué par cette méthode de protection. Le danger ne réside pas ici dans les risques d'infection, mais bien dans la capacité qu'a tout utilisateur légitime d'introduire un programme malveillant dans le système.

Un troisième problème, plus fondamental celui-là, se pose encore dans un contexte protégé tel que nous l'avons décrit. Le danger est le suivant : un virus suffisamment "intelligent" pourrait trouver la clé de déchiffrement d'un objet, ensuite obliger l'environnement de chiffrement à déchiffrer celui-ci, l'infecter, et finalement trouver la clé de chiffrement puis obliger l'environnement de chiffrement à le chiffrer à nouveau. Une attaque de ce type est la seule qui puisse rester indétectée par un "bon" schéma cryptographique. Rappelons que l'hypothèse est ici que tout objet est protégé et qu'il n'existe aucun moyen d'éviter le contrôle, même à titre exceptionnel. Nous verrons plus loin quels sont les risques que cette hypothèse nous épargne. Mais dans son champ d'application l'attaque décrite ci-dessus peut être contrée par l'utilisation d'un système cryptographique à clé publique comme proposé par POZZO et GRAY; dans ce cas la clé secrète n'est pas stockée dans le système, c'est à son propriétaire qu'il incombe d'en maintenir la confidentialité.

Dans cette méthode de protection nous avons interdit d'interpréter tout objet infecté, et pour cela nous avons proposé un schéma de détection. Un problème pratique se présente alors lors de la mise en oeuvre de cette méthode de détection. Par définition, elle doit pouvoir s'appliquer à tout objet du système, même (et surtout !) s'il est infecté; ceci signifie qu'il existe au moins un programme qui enfreint la règle de non interprétation d'un objet infecté : le programme de vérification lui-même. Or nous avons fait l'hypothèse qu'il n'existe aucun moyen de contourner le mécanisme de protec-

tion. Il en résulte un cercle vicieux interdisant la réalisation du programme de protection fonctionnant comme nous l'avons décrit. En conséquence, l'implémentation des mécanismes de détection et de suppression des objets infectés doit être réalisée par un module matériel par lequel transitent toutes les demandes d'accès à l'information, sans exception, comme présenté à la figure 6.4. L'accès ne sera autorisé qu'après vérification par ce matériel de l'intégrité de l'objet en cause. Dans le cas où une modification est détectée, l'accès est refusé et l'objet incriminé est physiquement effacé du disque avant d'être à nouveau chargé à partir d'une copie de sauvegarde saine.

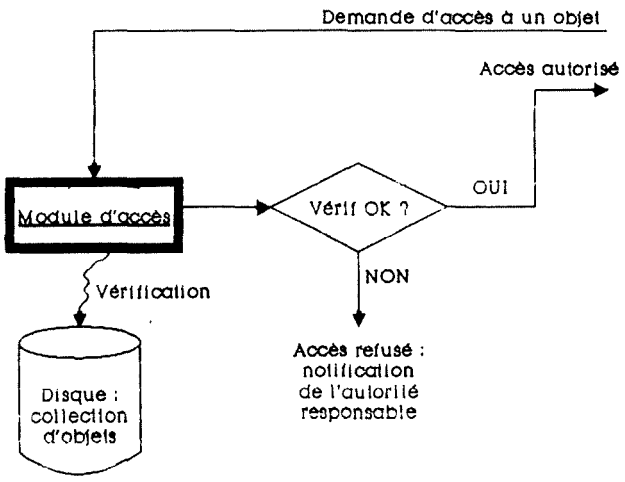

Figure 6.4. : Vérification d'intégrité par matériel

1.3. Conclusion

Nous avons vu que la seule méthode permettant de prévenir toute infection virale venue de l'extérieur est l'isolationnisme. Pour protéger univoquement un système autorisant un partage d'information avec d'autres systèmes, nous avons besoin d'une méthode permettant de détecter qu'un objet est infecté avant qu'il ne soit interprété. Cette détection sera alors suivie d'une restauration de l'objet infecté dans un état sain; cet objet ainsi désinfecté pourra alors être normalement utilisé.

La réalisation d'une telle méthode de détection peut être avantageusement réalisée en utilisant un système de chiffrement à clé publique pour signer les objets à protéger plutôt qu'en faisant appel à une technique conventionnelle à clé secrète. Si l'on désire faire l'hypothèse qu'il n'existe pas de moyen permettant de contourner la vérification, il faut alors

utiliser un module matériel pour réaliser le schéma.

2. Les méthodes non universelles

Les techniques de protection de l'information présentées à la section précédente comportent un certain nombre d'inconvénients. En effet, un utilisateur totalement isolé de l'extérieur ne fera jamais l'objet d'une contamination; de même un système limitant l'interprétation de l'information ou interdisant toute modification est également à l'abri. Ces solutions au problèmes des virus sont cependant inacceptables dans tout contexte scientifique ou de développement si l'on veut que celui-là ait une réelle utilité, ou si l'on désire bénéficier du travail d'autrui [COH-87]. L'objectif de cette section est de présenter des méthodes en général plus souples que celles déjà citées, mais sans perdre de vue qu'elles n'offrent pas une sécurité absolue.

Nous allons ainsi réexaminer les méthodes présentées précédemment, en les modifiant de telle sorte qu'elles offrent à présent un intérêt pratique. Nous exposerons ensuite les limites rencontrées par le concepteur d'un programme de protection contre les virus, et nous terminerons par quelques recommandations générales permettant d'éviter certains risques inutiles. Nous donnerons également un exemple de choix d'une protection dans un certain nombre d'environnements différents.

2.1. Limitation de la transitivité

Nous avons déjà dit qu'aucun système qui autorise un partage d'information pouvant être interprétée de manière générale n'est à l'abri d'une infection virale. Nous avons ajouté que l'infection avait pour domaine la fermeture transitive du partage d'information entre utilisateurs.

Cette seconde affirmation constitue une simplification de la pensée de COHEN. Nous allons à présent réparer cette inexactitude en disant que s'il n'y a pas de limitation sur la transitivité du flux d'information alors l'information peut atteindre la fermeture transitive du flux d'information en partant de n'importe quelle source [COH-87]. Cette absence de limitation sur la transitivité signifie que nous envisageons un système général dans lequel les utilisateurs sont capables d'utiliser l'information en leur possession

comme ils le désirent et de la passer à ceux qui en ont besoin.

Cette section a pour but d'examiner les méthodes permettant de limiter la transitivité du flux d'information dans le but d'empêcher la contamination totale d'un système une fois celui-ci contaminé. Certaines d'entre elles mènent à un partitionnement du système; ce sont celles que nous examinerons en premier lieu. Nous verrons ensuite des techniques moins catégoriques dans le principe.

2.1.1. Partitionnement du système

Le cloisonnement idéal doit réaliser une *partition* (4) du graphe des flux d'information, dont chaque sommet représente un utilisateur du système et dans lequel la présence de l'arc (i,j) signifie qu'un flux d'information est possible entre l'utilisateur i et l'utilisateur j.

Dans ce cas, aucune information ne peut être introduite dans une partie du système si elle provient d'une autre partie; l'infection éventuelle est limitée à la partie dans laquelle elle a lieu.

2.1.1.1. Modèles d'intégrité.

Ces modèles sont cités par COHEN [COH-87]; le lecteur intéressé pourra se référer aux publications originales [BEL-73, BIB-77]. Nous présentons brièvement les deux modèles, en montrant que leur utilisation simultanée conduit à un partitionnement du système.

2.1.1.1.1. Modèle de BELL-LaPADULA

Un *niveau de sécurité* est associé à chaque utilisateur; un utilisateur de niveau de sécurité donné ne peut pas lire de l'information appartenant à un utilisateur de niveau de sécurité supérieur au sien, ni écrire de l'information chez un utilisateur de niveau de sécurité inférieur au sien.

L'information tend ainsi à migrer vers les niveaux de sécurité supérieurs.

---

4) Une partition d'un graphe G est définie comme un ensemble de sous-graphes G_i non vides de G tels qu'il n'existe pas d'arc de G reliant un sommet de G et un sommet de $(G \backslash G_i)$ et que $(\bigcup G_i) = G$.

2.1.1.1.2. Modèle de BIBA

Il s'agit du dual du modèle de BELL-LaPADULA. Un *niveau d'intégrité* est ici associé à chaque objet; un objet de niveau d'intégrité donné ne peut pas être utilisé pour lire d'objet de niveau d'intégrité inférieur, ni pour écrire d'objet de niveau d'intégrité supérieur.

L'information tend ainsi à diminuer d'intégrité.

2.1.1.1.3. Conjonction de ces deux modèles

Lorsque l'on fait coexister les deux modèles, il en résulte un partitionnement du système en sous-systèmes fermés selon la transitivité, si par exemple les mêmes divisions sont utilisées pour les deux mécanismes (un niveau d'intégrité élevé correspondant à un niveau de sécurité élevé : 1). Dans ce cas, comme montré à la figure 6.5., un cloisonnement apparaît car l'information qui diminue d'intégrité (selon le modèle de BIBA elle a tendance à le faire) diminue également de niveau de sécurité; or c'est interdit par le modèle de BELL-LaPADULA. De même, l'information qui augmente de niveau de sécurité (selon le modèle de BELL-LaPADULA) augmente également d'intégrité, en contradiction avec le modèle de BIBA. Le cloisonnement est donc réalisé dans chaque niveau d'intégrité et, au sein d'un même niveau d'intégrité, dans chaque niveau de sécurité.

Huit autres stratégies de définition de la correspondance peuvent être appliquées. Dans l'exemple de la figure 6.5. l'infection peut seulement s'étendre depuis les niveaux d'intégrité élevés vers les niveaux plus bas à l'intérieur d'un même niveau de sécurité (cas où les limites du modèle de BIBA sont à l'intérieur de celles du modèle de BELL-LaPADULA : 2), ou en

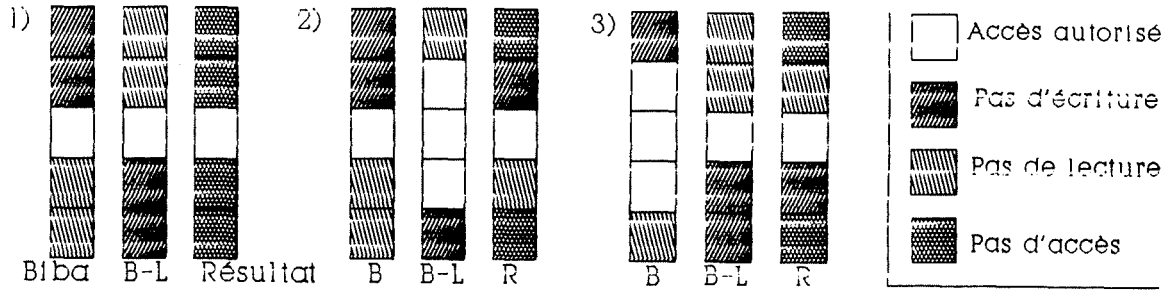

Figure 6.5. : Couplage des deux modèles

augmentant de niveau de sécurité au sein d'un niveau d'intégrité donné (limites du modèle de BELL-LaPADULA à l'intérieur de celles du modèle de BIBA : 3).

Remarquons que cette technique de partitionnement des systèmes a pour effet d'interdire à tout utilisateur d'écrire de l'information utilisable par tous les autres (en particulier, des programmes).

Un autre problème se pose lors de la mise en oeuvre d'une telle technique au sein d'un système réel. Il est en effet démontré qu'une méthode précise d'intégrité est la solution d'un problème NP-complet [DEN-82] et qu'il ne faut donc pas en attendre des performances raisonnables en temps dès que la taille des données atteint une certaine limite. Ce problème sera examiné par la suite.

L'intérêt de cette technique de protection est donc théorique : son implantation demanderait d'affaiblir le principe de cloisonnement absolu, ce qui aurait pour effet dans le meilleur des cas de ralentir l'infection, mais jamais de la limiter à un sous-ensemble du système.

2.1.1.2. Compartimentation

Une autre politique de cloisonnement est typiquement utilisée dans les applications de type militaire. L'ensemble des utilisateurs est ici partitionné en "compartiments" au sein desquels chacun n'est capable d'accéder qu'à l'information dont il a besoin. Comme chaque utilisateur n'a accès qu'à un seul compartiment à la fois, cette technique empêche l'infection d'un compartiment par un autre.

COHEN fait remarquer que les mises en oeuvre actuelles de ce partitionnement autorisent un utilisateur à accéder simultanément à plusieurs compartiments et que dans ce cas la contagion peut avoir lieu malgré le cloisonnement.

En résumé nous pouvons donc affirmer que ces deux méthodes de cloisonnement sont efficaces pour confiner la contagion au sous-système dans lequel elle a lieu, mais qu'elles sont parfois difficiles à mettre en oeuvre dans la pratique.

2.1.2. Limitation de la transitivité sans partitionnement

L'idée de limiter la contagion à une partie du système alors même qu'il n'est pas compartimenté semble a priori séduisante, mais on peut craindre intuitivement des difficultés de réalisation. Nous présentons ici une méthode générale de limitation de la transitivité; nous en examinerons ensuite un cas particulier présentant quelques avantages.

2.1.2.1. La liste des flux

Pour chaque objet, une liste de tous les utilisateurs ayant eu un effet sur cet objet est tenu à jour. La protection prend alors la forme d'une expression booléenne sur cette liste, et qui détermine l'accessibilité.

La gestion de ces listes est réalisée de la manière suivante: lors d'une action sur un objet, la liste de flux de l'objet résultat est constituée de la réunion des listes de flux de tous les objets utilisés pour le produire, auxquelles on ajoute l'utilisateur qui a requis l'action. Un exemple de gestion de listes est donné à la figure 6.6.

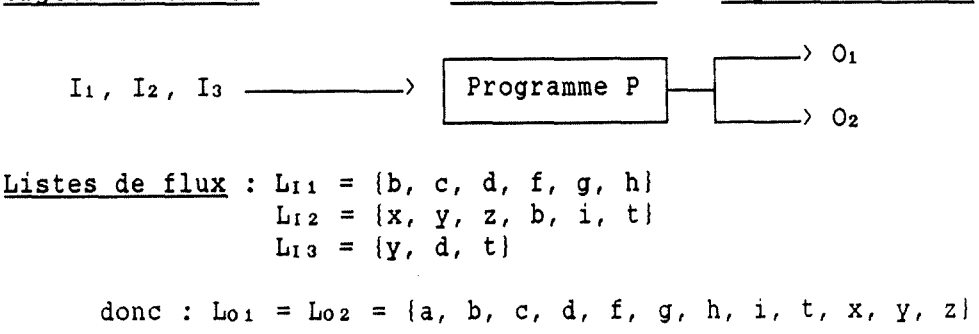

Figure 6.6. : Gestion de la liste de flux

Cette technique de limitation de la transitivité est très générale; en effet, la condition peut être définie de manière à reproduire une méthode déjà présentée. A titre d'exemple nous pouvons redéfinir la compartimentation de type militaire en disant qu'un utilisateur ne peut accéder qu'aux objets dont la liste de flux ne comporte que des utilisateurs du même compartiment que lui.

Une condition ne conduisant pas à un partitionnement du système serait, par exemple, que l'utilisateur X n'a le droit d'accéder à un objet que s'il a été écrit par les utilisateur A et B ou par les utilisateur A et C, mais pas par un seul d'entre eux. Cette stratégie a peut-être été adoptée par X parce qu'il considère A comme un programmeur compétent mais peu sûr, et qu'il n'est prudent d'utiliser son travail que lorsque celui-ci a été réalisé sous la "surveillance" de B ou de C.

2.1.2.2. Distance de flux

La politique de distance de flux est un cas particulier de la technique générale que nous venons d'exposer, convenant particulièrement bien pour la limitation de la contagion. Elle consiste à associer à chaque information la "distance" que celle-ci a déjà "parcourue" (nombre de partages), et à interdire que cette distance ne dépasse un maximum M fixé à l'avance. Un virus ne pourra alors infecter qu'un nombre limité d'utilisateurs, au maximum M-D_0 (D_0 est la distance déjà attachée à l'objet qui a contaminé le système lors de son infection).

Le calcul de cette distance est réalisé lors de chaque utilisation de l'information, en obéissant aux deux axiomes suivants :

- la distance du résultat est supérieure ou égale aux distances des entrées;
- la distance de l'information partagée est strictement supérieure à la distance de cette information avant le partage.

COHEN propose d'utiliser le maximum des distances des entrées comme distance du résultat, et la distance de l'information avant le partage augmentée de 1 comme distance de l'information partagée. Ainsi, un fichier de distance 8 partagé par un processus de distance 2 augmente la distance de ce processus à 9 (= max(8,2) + 1); chaque sortie du processus postérieure au partage sera au moins de distance 9.

Cet affaiblissement de la politique de listes de flux présente l'avantage sur les méthodes de partitionnement d'autoriser le partage d'information avec un ensemble d'utilisateurs non limité a priori. De plus, comme sa complexité est polynomiale et que la quantité de place qu'il nécessite n'est pas trop importante, il peut être implanté en pratique dans un système réel où il sera efficace.

Il nous faut cependant ici examiner un problème déjà cité et qui tempère les avantages de cette méthode de limitation de la contagion par mesure de la distance du flux d'information. Il s'agit d'une conséquence de la démonstration de DENNING [DEN-82] concernant la complexité NP-complète de la tenue à jour d'une liste de flux précise pour chaque objet partagé. En effet, toute solution d'un problème NP-complet mène à des performances médiocres dès qu'un certaine taille des données est atteinte. Ceci nous interdit l'utilisation d'une méthode de liste de flux précise pour limiter la contagion. Or, toute technique imprécise tend à transformer le système dans lequel elle est appliquée en un système au sein duquel tout utilisateur est isolé des autres [COH-87]. On postule en effet que ces méthodes utilisent des estimations prudentes des conséquences possibles d'une action. Or le système devient moins utilisable pour un utilisateur lorsqu'une information lui est injustement rendue inaccessible. Un tel système tend donc à devenir de moins en moins utilisable, jusqu'à ce qu'il devienne totalement isolationniste (si l'équilibre s'établissait ailleurs, le système serait précis puisqu'il n'évoluerait plus; cette précision requérant la solution d'une solution d'un problème NP-complet, toute solution de complexité polynomiale doit tendre vers un système sans partage d'information).

Pour conclure, nous dirons que ces méthodes de protection contre la contagion virale par limitation de la transitivité sont loin de constituer la panacée pour un système réel. En effet, la seule technique précise à la disposition du concepteur d'un système aux performances acceptables est de partitionner les utilisateurs, partitionnement qui est intolérable dans beaucoup d'environnements actuels mais qui protège efficacement de la contagion totale.

Des méthodes moins précises existent qui pourraient être utilisées dans un certain nombre de cas si l'on accepte leur tendance à l'isolationnisme.

Le concepteur se trouve donc en présence d'une alternative dont les deux branches se rejoignent à la limite.

2.2. Limitation de l'interprétation

Nous venons d'examiner quelques méthodes permettant de limiter la portée de la contagion au sein d'un système un fois celui-ci contaminé. Une démarche complémentaire peut être adoptée, qui consiste à tenter de freiner sinon d'interdire la contamination plutôt que d'essayer d'en réduire les conséquences.

En remarquant que l'infection vient de ce que l'information est interprétée comme un programme, nous avons déjà énoncé un moyen d'empêcher toute infection en interdisant d'interpréter toute information comme un programme. Un système fonctionnant de la sorte ne serait d'aucune utilité pratique; par conséquent, on pourrait essayer d'élargir cette restriction en remarquant qu'il est possible de définir un sous-ensemble d'opérations qui ne permettent pas à l'infection d'avoir lieu même dans le cas le plus général de partage et de transitivité. Certaines fonctions sont en effet indispensables pour réaliser une infection, par exemple la fonction WRITE. Mais comme un programme utile doit fournir des résultats, tout système présentant une utilité pratique doit donc pouvoir interpréter un ordre d'écriture. On ne sait pas s'il existe un tel sous-ensemble de fonctions qui ne serait pas *de premier ordre fixé*, c'est-à-dire qui aurait quelque utilité pratique [COH-87].

Mais dans tous les cas un système où l'interprétation n'est pas générale ne permet pas à la contagion de s'étendre plus loin que lors d'un schéma d'interprétation générale. Une borne supérieure à la contagion existe donc dans un contexte d'interprétation limitée; un tel contexte est ainsi préférable au schéma général lorsqu'il est applicable, dans un système de consultation de base de données, par exemple. Malheureusement, peu d'environnements actuels ne requièrent pas la généralité d'interprétation de l'information.

2.3. Signature - chiffrement

Dans un contexte où la généralité de l'interprétation est requise et où la transitivité du flux d'information ne peut pas être limitée, POZZO et GRAY [POZ-86, POZ-87] nous proposent une technique de détection des infections. Nous avons déjà présenté cette technique comme moyen de protection universel a posteriori; rappelons simplement qu'un mécanisme automatique réalise le déchiffrement de l'information chiffrée, ce qui permet de détecter toute

modification de cette information. Nous avions également élargi le champ d'application de cette méthode à toute information du système, en faisant l'hypothèse de généralité d'interprétation.

Dans le cadre d'une protection totale, cette méthode constitue une protection efficace si la désinfection est efficace. Cependant, la surcharge de travail imposée au système causera habituellement l'abandon de cette technique qui dégrade les performances.

Une solution intermédiaire est proposée par les auteurs, dans laquelle on ne prétend plus protéger universellement le système. La méthode exposée ici est donc un affaiblissement du schéma de signature universel. Il consiste simplement à définir quels sont les objets qu'il convient de protéger en priorité, et à permettre l'utilisation des autres sans protection.

Plusieurs problèmes apparaissent dès qu'une telle possibilité est introduite. Tout d'abord, il est évident qu'il faut trouver un moyen de faire savoir au mécanisme de validation si telle information doit être validée ou si elle doit être utilisée sans passer par lui. Un attribut supplémentaire est donc identifié pour tout objet, devant être géré par des routines additionnelles. La sécurité du système nécessite une protection à la fois de cet attribut supplémentaire et de ces routines afin d'éviter l'attaque simple, applicable dans un schéma de signature tel que nous l'avons décrit, consistant à marquer tout objet infecté comme ne nécessitant pas de validation. Cette attaque échoue bien entendu si l'on a adopté le schéma de chiffrement pur et simple des objets à protéger car elle serait immédiatement découverte lors du déchiffrement (les objets chiffrés ayant été modifiés).

Un autre problème apparaît lorsque certains objets sont protégés alors que d'autres ne le sont pas. En effet, lorsque plusieurs objets possèdent le même nom dans le système, le système d'exploitation utilise un mécanisme appelé *résolution* pour déterminer univoquement de quel objet il s'agit lors d'une référence à ce nom. Si un utilisateur anticipe mal ce mécanisme et qu'il invoque un objet par son nom, il y a un risque que ce soit à un autre objet que Le système d'exploitation accède. Indépendamment de la fausseté des résultats, le danger est que l'objet "résolu" par le système d'exploitation ne soit pas protégé alors que celui que l'utilisateur désirait invoquer l'était, ce qui a pour effet de diminuer à son insu la sécurité de cet utilisateur.

Afin de prendre en compte ces problèmes, POZZO et GRAY ont défini un schéma de gestion de risques (le *Risk Management Scheme* ou RMS). Les prin-

cipes du RMS sont les suivants : à chaque information est attachée un *niveau de crédibilité* et à tout processus est assigné un *niveau de risque*. La valeur de crédibilité d'un objet reflète la probabilité que cet objet est infecté et est déterminée par l'administrateur du système; plus l'information est "sûre", plus sa valeur de crédibilité est élevée. Le niveau de risque d'un processus est *hérité* du processus père ou *assigné* par l'utilisateur qui le crée; il reflète le degré de sécurité que l'utilisateur désire maintenir : si son niveau de risque est haut, la probabilité d'une contamination est basse pour lui.

Ces niveaux de crédibilité et de risque sont mis en correspondance par le système d'exploitation, qui est responsable d'empêcher un processus de niveau de risque donné d'utiliser un objet de niveau de crédibilité inférieur.

Une dérogation à ce principe est prévue, permettant à un processus d'utiliser des objets de niveaux de crédibilité inférieurs à son niveau de risque; dans cette optique, des commandes spéciales permettent de contourner la protection mise en oeuvre par le schéma, au prix d'une perte de sécurité. L'avantage de procéder ainsi est de rendre explicite cette perte de sécurité par l'emploi de commandes de nom différent de ceux des commandes standard (par exemple, RUN et RUN_UNTRUSTED pour les commandes de lancement d'un programme).

Le problème majeur posé par ce schéma est de nouveau l'obligation de protéger ces nouveaux attributs des objets et des processus que sont les niveaux de crédibilité et de risque.

En résumé, cette méthode consiste en un mécanisme de validation partielle dans lequel on peut définir des objets "critiques" à protéger. La coexistence d'objets validables et d'objet non protégés pose un certain nombre de problèmes, auxquels le RMS peut apporter des solutions.

2.4. Programmes de protection

Nous poursuivons l'examen des moyens de protection contre les virus par un rapide survol de quelques progiciels conçus dans le but d'offrir un accroissement de sécurité à leurs utilisateurs ([5]); une évaluation de la

---

[5]) Les progiciels présentés ici sont en général destinés à un environnement P.C., mises à part quelques exceptions.

sécurité réellement apportée par de telles méthodes sera ensuite développee.

2.4.1. Protection a priori

Les progiciels qui offrent une protection a priori du système sont de trois types; certains permettent une vérification du code d'un programme, d'autres fonctionnent comme des filtres qui avertissent l'utilisateur des comportements "dangereux" du programme en cours d'exécution, et les derniers que nous avons relevés réalisent un contrôle des droits d'accès aux fichiers.

2.4.1.1. Vérification de code suspect

Le progiciel CHECK-4-BOMB, de SWARTHMORE SYSTEMS, tente de vérifier si un programme, dont le nom lui est passé par l'utilisateur, est potentiellement dangereux. Il fournit ainsi un relevé des instructions "sensibles" que ce programme contient, comme l'instruction qui permet sous MS-DOS de formater une piste. Il produit également un listing de toutes les chaines ASCII contenues dans le programme, ce qui permet de visualiser des messages du type "Arf, Arf ! I got you", typiques à certains programmes "malfaisants".

Ce progiciel est donc plutôt conçu dans un but de protection contre les chevaux de Troie; il ne détectera un virus que dans la mesure où celui-ci comporte un déclenchement suffisamment "agressif".

2.4.1.2. Filtres

La plupart des filtres anti-virus fonctionnent comme des tâches en arrière-plan ("background") ou comme un gestionnaire de périphérique ("driver de disque") qui intercepte les actions jugées dangereuses. Certains interdisent l'action en question, d'autres demandent une confirmation de l'utilisateur avant de l'autoriser.

Par exemple, le progiciel DPROTECT, de GEE WIZ SOFTWARE COMPANY, protège une disquette ou un disque dur contre toute écriture jusqu'au prochain démarrage du système. En cas de tentative d'écriture, une réinitialisation matérielle ("cold reboot") est exécutée, empêchant ainsi toute écriture sur le médium protégé.

Une protection par programme résident est efficace tant que la table du système d'exploitation qui contient l'adresse de ce programme est elle-

même protégée. Ainsi, dans l'exemple de la figure 6.7., le progiciel intercepte dans un premier temps toute demande d'écriture; il effectue sa vérification, puis passe la main au gestionnaire habituel. Après le détournement de la protection, les ordres d'écriture ne seront plus filtrés : ils sont directement adressés au gestionnaire des écritures. Ce détournement peut être simplement réalisé en remplaçant dans une table (appelée *vecteur d'interruptions* sous MS-DOS) l'adresse du programme résident (en l'occurrence, DPROTECT) par celle du gestionnaire habituel. Il est à remarquer qu'aucune protection de l'intégrité de cette table n'est proposée par MS-DOS.

D'autres progiciels fonctionnent selon le même principe, comme par exemple ANTIDOTE, de QUAID SOFTWARE Ltd.; BOMB SQUAD, de SWARTHMORE SYSTEMS; C-4, de INTERPATH Corporation; FLU SHOT 3, de Ross GREENBURG.

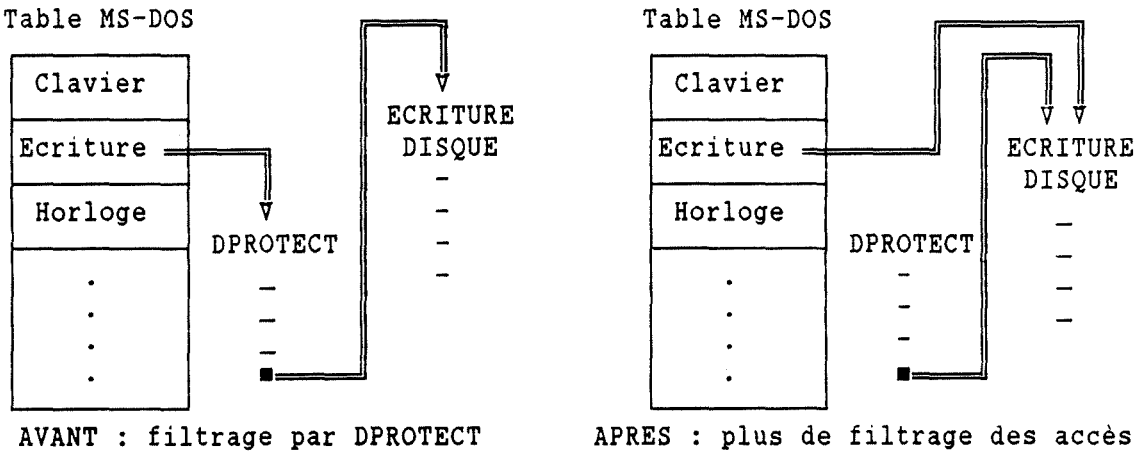

Figure 6.7. : Détournement de la protection logicielle

2.4.1.3. Contrôle des droits aux fichiers

Un progiciel réalise sur P.C. l'équivalent du système de protection des systèmes d'exploitation gérant un système multi-utilisateurs (on assigne des attributs protégés aux fichiers; ces attributs sont choisis dans l'ensemble Read, Write, Execute, Delete). Ce progiciel est FLU SHOT PLUS, de Ross GREENBURG.

2.4.2. Protection a posteriori

Nous avons relevé trois types de progiciels qui offrent une protection a posteriori; les premiers proposent un contrôle d'intégrité, d'autres constituent une aide dans la gestion des copies de sauvegarde, et les derniers se proposent de désinfecter un système contaminé.

2.4.2.1. Contrôle d'intégrité

Certains des programmes réalisent un contrôle d'intégrité constituant une version affaiblie du schéma proposé par POZZO et GRAY. Il s'agit par exemple de VI-RAID, de PRIME FACTORS; du module SURGICAL de Dr PANDA UTILITIES, de PANDA SYSTEMS; de VACCINE, de SOPHOS Ltd.

D'autres progiciels permettent au programme auquel ils sont liés de s'auto-tester en préalable à chaque exécution. Nous avons connaissance de ANTIGEN, de DIGITAL DISPATCH Inc., et de CRYPTOGRAPHIC CHECKSUM, de Fred COHEN.

2.4.2.2. Gestion des copies de sauvegarde

Les progiciels auxquels nous pensons ici ne sont pas conçus dans le but de se protéger contre les virus, mais ils peuvent y contribuer lorsqu'ils sont utilisés pour restaurer un système infecté dans un état sain. Nous citerons seulement l'utilitaire BACKUP, de MICROSOFT Corporation, qui permet de réaliser la gestion des copies de sauvegarde.

2.4.2.3. Désinfection d'un système contaminé

Certains progiciels sont censés être capables de supprimer d'un objet infecté le code du virus qui s'y trouve, de manière à le remettre dans son état original.

Par exemple, les programmes ANTIGEN et DATA PHYSICIAN de DIGITAL DISPATCH Inc. tentent de réaliser cette prouesse.

2.4.3. Evaluation

Après avoir établi une liste de certains produits logiciels dont le but est d'offrir une protection contre les attaques virales, nous allons à présent examiner le degré de sécurité qu'un tel programme est capable d'apporter. Comme nous n'avons réalisé aucun test sur les progiciels présentés (leur description est basée sur les affirmations de leurs constructeurs [HIG-88]), les seules critiques que nous pouvons émettre se situent au niveau des principes théoriques de ces programmes.

Nous allons tout d'abord démontrer un résultat intéressant qui fait la preuve que la détection des virus est indécidable [COH-87]. Il est en effet impossible de construire un programme D qui se termine toujours, qui ne fournit jamais de résultat faux et qui répond par "oui" ou par "non" à la question suivante : "le programme P que voici est-il un virus ?", quel que soit ce programme P qui lui est passé en paramètre.

Pour le démontrer, supposons qu'un tel programme D existe, qu'il se termine toujours et qu'il fournit la réponse "oui" dans le cas où le programme P est un virus, "non" sinon, sans jamais se tromper.

On peut alors construire le programme V de la figure 6.8., qui fait exécuter D sur lui-même et teste la réponse fournie lors de la terminaison de l'appel D(V). Il suffit alors à V d'agir de manière à contrarier systématiquement cette réponse : lorsque D a détecté que V était un virus alors V infectera un autre programme, et V ne fera rien si D a répondu "oui".

```
SI D(V) = "non"
ALORS
      REPETER
            Fich <- choisir_un_programme_au_hasard;
      JUSQU'A (pas_encore_infecte (Fich));
      ajout_virus_au_fichier (Fich);
      aller_a (debut_programme_infecte);
SINON
      ne_rien_faire;
```

Figure 6.8. : Virus indécidable

La construction du virus V nous montre qu'il existe au moins un programme qui, lorsque D lui est appliqué, lui fait fournir systématiquement une réponse fausse si D s'est terminé. Nous pouvons en déduire que le programme D tel qu'il est spécifié plus haut est impossible à construire; il nous faut

donc admettre qu'un programme conçu dans ce but ne se termine pas dans tous le cas (par exemple en bouclant indéfiniment) ou, s'il se termine dans tous les cas, fournisse parfois une réponse fausse. Par conséquent, l'indécidabilité de la détection des virus est démontrée.

Ce résultat nous montre immédiatement que les progiciels dont le but est la désinfection des programmes infectés ne peuvent pas fonctionner dans tous les cas. En supposant qu'ils se terminent toujours (ce qui est probable pour des produits commerciaux) il existe alors des virus qui passeront à travers les mailles du filet et/ou des programmes qui seront désinfectés "à tort", aussi sophistiqués que soient les algorithmes de détection utilisés.

Il est par contre possible de construire un antidote permettant de désinfecter tout objet infecté par un virus donné, et le résultat précédent nous montre qu'il est également possible d'améliorer ce virus (par exemple en le faisant évoluer) pour contourner la protection offerte par l'antidote. Ce phénomène a pour conséquence une situation de survie du plus intelligent analogue à la situation biologique [COH-88].

COHEN nous fait de plus remarquer que, même si un programme de désinfection capable d'éliminer le virus infectant un système à un moment donné existe, il ne suffit pas de le lancer pour éliminer totalement ce virus du système. Il faut de plus que cet antidote ait accès aux copies de sauvegarde réalisées depuis la contamination du système, et que la désinfection ait lieu lorsque tout service est refusé à tout utilisateur. En effet, si ces conditions ne sont pas réunies, une recontamination peut avoir lieu lorsque le système sera remis en contact avec une copie de sauvegarde contaminée (qui a ainsi constitué un moyen "d'hibernation" du virus), ou la décontamination peut même ne jamais se terminer si le virus continuait à se reproduire pendant que le programme de désinfection opère; un objet désinfecté pourrait ainsi être immédiatement réinfecté. Il faut donc en quelque sorte désinfecter l'environnement du système comme on désinfecterait les draps d'un malade, et "donner la fièvre" au système (le figer) pour qu'il refuse tout service tant que l'infection n'est pas éliminée.

Pour conclure, nous pouvons dire que ces progiciels de désinfection peuvent avoir une utilité dans certains cas s'ils sont bien conçus mais qu'il serait dangereux de leur attribuer des vertus qu'ils ne peuvent pas posséder en basant sur leur présence un sentiment euphorique de sécurité.

Un autre type de progiciels peut être également la cause d'un fallacieux sentiment de sécurité totale; l'adoption d'un schéma de protection

par chiffrement, signature ou calcul de "checksum", même s'il apporte un accroissement de la sécurité, n'offrira une protection totale que dans certains cas bien précis. Dans la majorité des cas, l'adoption d'une telle technique rendra certes beaucoup plus difficile la conception d'un virus indétecté mais ne pourra pas l'interdire [COH-88].

En résumé nous pouvons donc affirmer que l'utilisation d'un progiciel de détection des virus accompagnée d'une désinfection des objets infectés peut être efficace dans certains cas; malheureusement, ces mesures seront inopérantes face à un virus suffisamment intelligent.

2.5. Quelques recommandations

Nous terminerons l'exposé des méthodes non universelles de protection contre les virus par quelques recommandations pouvant sensiblement améliorer la capacité de résistance des systèmes à l'infection.

La première de ces recommandations est d'opter pour une méthode de protection parmi celles que nous avons exposées, ou pour toute autre pourvu qu'elle soit efficace. Le choix d'une méthode résultera d'un compromis entre son coût (financier et en terme de perte de performances) et le niveau de sécurité qu'elle permet d'atteindre.

Par mesure de précaution, dans un système non universellement sûr, il serait préférable de faire inspecter les sources de tout nouveau programme par une équipe qui ne l'a pas écrit; les sources ainsi validées pourront alors être compilées sur place [FAK-88]. Cette manière de procéder décourage de plus la programmation non structurée ("tricky programming") propice à cacher des tentatives de malveillance ou plus simplement des erreurs.

Tout programme indisponible en code source (suite par exemple à un refus de la part du vendeur de les livrer) devrait provenir d'un endroit dans lequel on peut avoir confiance et qui dispose lui-même d'un protection offrant des garanties suffisantes (6).

On recommande également d'éduquer les utilisateurs à une grande circonspection concernant les importations de programmes inédits dans le système; de meme les super-utilisateurs du système doivent être conscients du

---

6) On a en effet vu des logiciels infectés au sein des firmes qui le développaient, à l'insu de celles-ci, et vendus tels quels (voir [COH-88] p.169, ou [ELM-88] p.42, par exemple).

risque de contagion que leur présence introduit et se connecter de préférence avec les privilèges d'un utilisateur normal à chaque fois que c'est possible.

Finalement, un cloisonnement maximal peut permettre d'éliminer l'infection avant qu'elle n'aie causé de dommages irréparables. En particulier, les environnements "à risques" (qui partagent souvent de l'information avec l'extérieur) devraient constituer une partie isolée du système.

2.6. Conclusion

Trois types de méthodes non universelles de protection anti-virale ont été relevés :

- les limitations de la transitivité du flux d'information pouvant être réalisées en pratique mènent à l'isolationnisme, soit directement lorsqu'elles ont la forme d'un cloisonnement du système, soit après un certain temps s'il s'agit de méthodes imprécises;
- les restrictions sur la généralité de l'interprétation de l'information attendent encore une étude théorique plus poussée qui permettra peut-être de les généraliser à des systèmes non fixés;
- le schéma d'intégrité de POZZO et GRAY peut être appliqué dans un système réel pour lui apporter une nette amélioration de son degré de sécurité, à condition que des choix judicieux des paramètres aient été effectués. Un progiciel de protection peut être utile dans certains cas mais il doit être choisi avec soin.

La sécurité offerte par ces protections sera sans nul doute accrue par l'adoption de certaines règles de prudence. Mais il n'en reste pas moins qu'un virus suffisamment intelligent pourra déjouer tous ces pièges; sa conception en est cependant rendue beaucoup plus ardue.

3. Proposition de solutions

La recherche d'une solution pour protéger un système contre les virus passe par la définition du degré de sécurité désiré dans ce système. Cette définition doit également stipuler quel niveau de performances minimal doit être atteint. C'est la conjonction de ces objectifs en général contradictoires qui déterminera le choix d'une solution particulière. A titre d'exemple, nous examinons ici quelle solution semble devoir être retenue dans

quelques environnements typiques.

3.1. Environnement partagé de haute sécurité

Lorsque l'on est en présence d'un environnement partagé très sensible, où le coût financier ne représente pas une contrainte (budget quasi-illimité) mais où la protection doit être maximale, il convient d'utiliser un schéma de protection du type POZZO et GRAY, appliqué à la totalité des objets du système. Le niveau de performances désiré pourra alors être atteint en ajoutant au système des coprocesseurs de chiffrement spécialisés (par exemple DES pour le calcul du condensé du fichier, et RSA pour la signature). Comme ce schéma doit également être assorti de mesures de restauration du système en cas de détection d'une infection, il faudrait y ajouter par exemple un sous-schéma de sauvegarde continue qui a pour but de conserver les N dernières versions de chaque objet. Cette sauvegarde doit être réalisée de manière à empêcher toute modification d'un objet archivé en effectuant le stockage sur un support irréversiblement inscriptible (comme un disque numérique ineffaçable, par exemple). Une procédure de récupération de la dernière version archivée doit alors être prévue, afin de restaurer le système dans l'état sain qui était le sien avant l'infection.

Ce schéma possède en outre l'avantage de protéger efficacement le système contre des pertes d'information dues à un "crash" des disques ou à toute autre cause.

3.2. Environnement P.C.

Dans un environnement P.C., le coût financier doit être maintenu à un niveau acceptable (pas plus de 10 % de la valeur du matériel, par exemple, ce qui constitue un budget relativement limité compte tenu de la tendance à la baisse des prix). Il semble alors qu'une version affaiblie de l'isolationnisme représente la meilleure solution. On évitera donc tout contact direct avec l'extérieur, en restant très prudent dans le choix des fournisseurs de nouveaux programmes; il convient également d'inspecter dans la mesure du possible les sources de ces programmes, éventuellement d'y inclure un module auto-vérificateur d'intégrité du type "cryptographic checksum", et surtout de conserver des copies de sauvegarde d'un nombre suffisant de versions des

objets modifiés (bien entendu, il est indispensable de les conserver en un endroit où aucun virus ne peut avoir accès !).

3.3. Environnement "commercial" partagé

Pour terminer, nous envisagerons un environnement plus classique qui se situe entre les deux contextes qui viennent d'être évoqués. Il s'agit d'un système partagé où on désire une sécurité élevée pour un coût le plus bas possible (financier et en termes de dégradation des performances).

En examinant ce qui était coûteux dans la solution la plus sûre proposée ci-dessus, nous remarquons les coprocesseurs spécialisés (de plus bien souvent soumis à une autorisation du *Department of Defense* s'il s'agit de produits américains) ainsi que le système de sauvegarde continue. Les premières pourront être remplacées par un simple coprocesseur mathématique si on accepte une baisse de performances. Si une telle baisse est inacceptable, un schéma de type RMS devra être mis en oeuvre, aux dépens de la sécurité. Quant aux sauvegardes, elles pourraient être effectuées moins souvent, au prix d'une éventuelle perte d'information lors d'une restauration du système dans un état sain.

4. Conclusion

Nous avons exposé dans ce chapitre les principales méthodes de protection anti-virales. Nous avons tout d'abord examiné les techniques permettant d'empêcher toute contamination; leur emploi est malheureusement limité à un nombre restreint de systèmes. D'autres techniques permettent l'élimination de la contamination avant que toute contagion puisse avoir lieu; elles présentent l'inconvénient de diminuer sensiblement les performances du système.

Des méthodes partielles ont ensuite été abordées. Un partitionnement du système peut limiter efficacement la contagion lorsque l'on se trouve dans un contexte où c'est applicable. Par contre, l'implantation d'autres techniques de limitation des flux semble impraticable à cause de certains problèmes de complexité. Un schéma RMS peut amener un "plus" au niveau de la sécurité; la difficulté réside ici dans la détermination judicieuse des paramètres.

Des programmes de protection actuellement disponibles ont été cités à titre d'exemple, mais leur efficacité doit encore être démontrée.

Finalement, tout système quel qu'il soit peut voir sa sécurité améliorée par l'adoption de certaines propositions relevant principalement du bon sens.

L'illustration du choix d'une de ces méthodes a permis de montrer qu'il faut analyser les solutions existantes en fonction des objectifs contradictoires que sont, dans le domaine de la sécurité comme dans la vie économique, le moindre coût pour un maximum de profit.

CONCLUSION

CONCLUSION

La protection des logiciels est l'un des multiples aspects du vaste domaine de la sécurité informatique. L'apparition de la malveillance a engendré le besoin de protéger les logiciels contre les manipulations; la croissance continuelle de ce phénomène renforce cette nécessité chaque jour davantage.

Une nouvelle menace est apparue avec les virus informatiques car il s'agit d'une forme de malveillance qui ne nécessite pas l'exploitation de faiblesses du système; leur élimination en est rendue très difficile dans la pratique.

Ce mémoire avait pour but de montrer l'importance de ces problèmes et de tenter d'y apporter quelques voies de solutions.

Nous avons dégagé deux philosophies pour nous protéger de la malveillance contre les logiciels. La solution humaine semble un peu délaissée à l'heure actuelle, mais les horizons ouverts par les protections légales, socio-organisationnelles et déontologiques sont prometteurs. La malveillance informatique étant avant tout un problème humain, il convient de la combattre par des moyens humains.

Ces méthodes ne permettront cependant pas d'éliminer totalement la malveillance envers les systèmes informatiques.

Un deuxième axe de protection peut alors être envisagé : mettre en oeuvre des solutions apportées par la technique pour suppléer aux lacunes des solutions humaines.

Quatre niveaux de protections techniques ont été identifiés. Le premier est mis en oeuvre par l'utilisateur légitime d'un logiciel qui désire disposer de moyens lui permettant de conserver l'exclusivité de la lecture du code du logiciel. En protégeant ainsi la confidentialité de ses programmes, il protège également son droit de propriété.

Deux réalisations de ce principe ont été présentées : le chiffrement du code, permettant de reporter la confidentialité du logiciel sur celle d'une clé, et la fragmentation-dissémination dont l'avantage est de maximiser la disponibilité du logiciel tout en minimisant les points durs de sécurité.

Le distributeur du logiciel désire lui aussi conserver l'exclusivité d'un droit; c'est le deuxième niveau de protection des logiciels. Il s'agit

ici d'empêcher les copies non autorisées du produit vendu, afin de garantir le revenu du fabriquant. Ce problème a pris un essor considérable avec l'apparition des ordinateurs personnels, les particuliers étant beaucoup plus enclins que les organisations à se livrer au "piratage" de logiciels.

Les distributeurs se sont dotés de moyens de protection de plus en plus sophistiqués, mais ceux-ci ont massivement fait l'objet de piratages systématisés, notamment au moyen de progiciels qui permettent la copie de l'image physique d'une disquette. La protection par un élément externe, liant le logiciel à un constituant matériel plutôt qu'au support, est plus efficace mais aussi plus coûteuse. La solution du mode de distribution permet dans certains cas une protection efficace. Pour un environnement plus "public", le contrôle de l'existence d'un droit semble le meilleur choix.

Le troisième niveau de protection est appliqué par le possesseur légitime du logiciel. Il veut être certain que le programme qu'il exécute n'a pas été modifié depuis son acquisition, afin de garantir les résultats de son travail. La protection de l'intégrité est réalisée soit par interdiction des modifications, soit par détection des changements illicites.

L'immuabilité du logiciel n'est pas applicable dans tous les cas mais elle offre de bonnes garanties lorsque l'environnement permet son utilisation. La détection des changements illicites fait appel aux techniques cryptographiques; elle offre une sécurité appréciable dans la plupart des cas. Le choix de l'algorithme et des protocoles de gestion est fondamental pour cette solution car elle conditionne son efficacité.

Le dernier niveau de protection technique des logiciels que nous avons envisagé est utilisé lorsque l'on désire réglementer l'accès à certains programmes. La plupart des systèmes d'exploitation offrent ce type de service; certains logiciels se chargent de surcroit de contrôler eux-mêmes les personnes qui tentent de les exécuter. Ces contrôles d'accès supposent comme préalable l'identification de la personne qui demande l'accès; cette identification se base sur quelque chose que l'utilisateur est, qu'il connait ou qu'il possède.

Dans cette dernière catégorie, le "Maitre-Achat" est probablement la carte à microprocesseur, qui est disponible à un coût raisonnable tout en offrant des avantages incomparables pour la sécurité.

Nous nous sommes également attaché à rechercher des solutions au problème des virus informatiques. Quelques méthodes de protection totalement sûres existent, mais elles sont malheureusement inapplicables en pratique.

D'autres methodes moins radicales ont alors été presentées.

La limitation de la transitivité peut s'appliquer à certains cas, mais les techniques les plus sûres mènent à un partitionnement du système. La limitation de l'interprétation pourrait également être efficace; des études doivent encore être réalisées dans ce domaine afin de démontrer si un système ainsi limité peut avoir une utilité pratique.

Plutôt que de tenter d'empêcher l'infection, on peut essayer de la détecter avant que des dégâts aient pu être commis. C'est selon ce principe que fonctionne le schéma de détection de POZZO et GRAY, en utilisant des méthodes cryptographiques pour réaliser une signature des objets à protéger.

Nous avons également examiné le principe théorique de certains progiciels de protection actuellement sur le marché. Une protection purement logicielle semble peu crédible; néanmoins, les programmes de gestion des copies de sauvegarde ou de vérification d'intégrité peuvent apporter un accroissement notable de la sécurité. La désinfection automatique semble par contre nettement plus hasardeuse lorsque le virus est inconnu du progiciel.

L'illustration du choix d'une méthode de protection a été donnée pour terminer, mise en oeuvre dans trois types d'environnement classiques. Ces trois exemples ont permis de montrer que les solutions possibles doivent être analysées en tenant compte des objectifs et des contraintes, à savoir d'une part la sécurité et les performances, et d'autre part leur coût.

Cette remarque peut s'appliquer à l'ensemble des solutions permettant de garantir la sécurité informatique. Telle solution offre une protection excellente, mais son coût est prohibitif; telle autre est très facile à mettre en oeuvre, son coût est faible et ses performances honorables mais la sécurité offerte est insuffisante dans le contexte où on désire l'appliquer.

L'enseignement à retirer de ce mémoire est donc que LA méthode de protection des logiciels n'existe pas, que la sécurité ne pourra jamais être assurée de manière absolue et que le choix d'un moyen de protection est à effectuer de manière raisonnée, en fonction des objectifs mais aussi des contraintes du système à protéger.

ANNEXE

ANNEXE 1 : UN PSEUDO-LANGAGE DE REPRESENTATION D'ALGORITHMES

Ce pseudo-langage a été utilisé pour la définition et la description de quelques algorithmes de virus.

1. Sémantique du langage de définition

Les symboles ...	Signifient ...
‹X›	le mot X est utilisé dans la définition de la syntaxe du pseudo-langage de représentation d'algorithmes
X ::= Y	X est défini par Y
{X}	X peut apparaître un nombre quelconque de fois (y compris 0)
X | Y	X ou Y
"X"	le mot X est un mot réservé du pseudo-langage

2. Syntaxe du pseudo-langage de représentation d'algorithmes

‹algorithme› ::= ‹instruction› {‹instruction›} |
‹definition_de_procedure›

‹instruction› ::= ‹instruction_simple› | ‹instruction_composee›

‹instruction_simple› ::= ‹variable› "‹-" ‹expression› ";"

‹variable› ::= ‹lettre› {‹caractere›}

‹caractere› ::= ‹lettre› | ‹chiffre›

‹lettre› ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z" | "_"

‹chiffre› ::= "0" | "1" | ... | "9"

‹expression› ::= ‹variable› {‹operateur› ‹variable›}

‹operateur› ::= "+" | "-" | "*" | "/"

‹instruction_composee› ::= ‹repeter_jusqu_a› | ‹tant_que›
| ‹si_alors› | ‹si_alors_sinon›
| ‹appel_de_procedure› ";"

```
<repeter_jusqu_a> ::=   "REPETER"
                              <corps>
                        "JUSQU'A (" <condition> ");"

<tant_que> ::=    "TANT QUE (" <condition> ") FAIRE"
                        <corps>

<si_alors> ::=    "SI (" <condition> )"
                  "ALORS" <corps>

<si_alors_sinon> ::=    "SI (" <condition> )"
                        "ALORS" <corps>
                        "SINON" <corps>

<corps> ::= <instruction> {<instruction>}

<condition> ::=   <expression> <connecteur> <expression> |
                  <expression> <connecteur> <valeur_booleenne> |
                  <valeur_booleenne>

<connecteur> ::= "<" | ">" | "=" | "≤" | "≥" | "≠"

<valeur_booleenne> ::= "FAUX" | "VRAI"

<appel_de_procedure> ::= <en_tete_de_procedure>

<en_tete_de_procedure> ::= <nom_de_procedure> {"(" <parametres> ")"}

<parametres> ::= <param> {"," <param>}

<param> ::= <expression> | <appel_de_procedure>

<nom_de_procedure> ::= <lettre> {<caractere>}

<definition_de procedure> ::= "PROCEDURE" <en_tete_de_procedure> ";"
                                    <algorithme>
```

3. Sémantique du pseudo-langage

- Un appel de procédure a pour effet l'exécution du sous-programme nommé. La valeur d'initialisation des paramètres est la valeur de l'expression figurant dans l'appel. La valeur de retour des paramètres est leur valeur en fin de procédure.
- Une affectation (<instruction_simple>) a pour effet d'affecter à la variable de gauche la valeur de l'expression de droite.
- L'évaluation d'une expression se fait de gauche à droite, séquentiellement.
- L'instruction "REPETER ... JUSQU'A" a pour effet l'exécution complète des instructions qui figurent dans son <corps>, le test de la <condition> et le rebouclage au début de la première instruction si la <condition> est fausse.

On peut encore définir son exécution de la manière suivante :

```
PROCEDURE REPETER_JUSQU'A (<condition>, <corps>) :
      <corps>;
      SI <condition> = FAUX
      ALORS REPETER_JUSQU'A (<condition>, <corps>);
```

- L'instruction "TANT QUE ..." a pour effet le test de la <condition>, l'exécution du <corps> si la <condition> est vraie et enfin le rebouclage en début de test de la <condition>.

  On peut encore le définir de la manière suivante :

```
PROCEDURE TANT_QUE (<condition>, <corps>) :
      SI <condition> = "VRAI"
      ALORS <corps>;
            TANT_QUE (<condition>, <corps>);
```

4. Procédures prédéfinies

nom (paramètres)	Spécifications
ajout_ligne_suivante_du_virus (F)	Ajout la ligne courante du virus à la fin du fichier F.
ajout_mot (MOT, F)	Ajoute le mot MOT à la fin du fichier F.
ajout_virus_au_fichier (F)	A l'issue de cette procédure, le programme F est infecté (il contient donc le code du virus, de sorte qu'on ait : pas_encore_infecte(F) = FAUX).
aller_a (ADR)	Cette procédure a pour effet de faire continuer l'exécution du programme à l'adresse ADR.
bit_au_hasard	Renvoie la valeur 0 ou la valeur 1 de manière imprévisible.
choisir_un_programme_au_hasard	Renvoie le nom d'un programme quelconque mais qui existe dans le système.
condition_de_declenchement	Renvoie la valeur VRAI si la condition de déclenchement prévue par le concepteur du virus est remplie au moment de l'appel, FAUX sinon.
debut_programme_infecte	Renvoie l'adresse de début du programme infecté qui se trouve en mémoire (celui qui contient le code du virus en cours d'exécution).

fin_virus	Renvoie la valeur VRAI si la fin du virus est atteinte (par exemple, fin de fichier), FAUX sinon.
ne_rien_faire	L'exécution de cette procédure n'a aucun effet.
operateur_au_hasard	Renvoie un des quatre opérateurs, choisi de manière aléatoire.
pas_encore_infecte (F)	Renvoie la valeur VRAI si le programme F n'est pas encore infecté, FAUX sinon.
variable_au_hasard	Renvoie un nom de variable syntaxiquement correct.

BIBLIOGRAPHIE

REFERENCES BIBLIOGRAPHIQUES

[ACH-86] : ACHEMLAL, M., MOURIER, M., *Dynamic Signature Verification*, in *IFIP/SEC'86*, Monte-Carlo, 1986.

[ACM-89] : numéro des *Communications of the ACM* consacré au "worm" d'ARPANET, juin 1989.

[AKL-83] : AKL, S.G., *On the Security of Compressed Encoding*, in *Advances in Cryptology, Proceedings of Crypto 83*, Plenum Press, New-York, 1984, p. 209-230 (cité dans [GIL-88]).

[ANO-86] : Anonyme, *Archivage électronique*, in *Séminaire CAP SOGETI INFORMATION*, Paris, 1986.

[ANO-88] : Anonyme, *The Seven Myths of Data Security*, Racal-Guardata, Fleet (England), 1988.

[APS-86] : APSAIRD, *Enquête et statistiques nationales sur la sécurité informatique*, APSAIRD, 1986, cité dans [ILI-88] et dans [LAM-88].

[BAU-87] : BAUVAL, A., CAMPANA, M., DUDET, M., GIRAULT, M., PAILLES, J-C, *Cryptographie et sécurité*, in *Bulletin de liaison de la recherche en Informatique et en Automatique*, n° 114, INRIA, Rocquencourt, 1987, p. 6-19.

[BEL-73] : BELL, D.E., LaPADULA, L.J., *Secure Computer Systems : Mathematical Foundations and Models*, in *The MITRE Corporation*, 1973.

[BIB-77] : BIBA, K.J., *Integrity Considerations for Secure Computer Systems*, USAF Electronic Systems Division, 1977.

[BUR-86] : BURTONBOY, G., *Virologie générale*, Centre d'impression bénévole, Cercle Médical Saint-Luc, UCL, 1985-1986.

[CAM-87] : CAMION, P., *Le projet CODES et la sécurite informatique*, in *Bulletin de liaison de la recherche en Informatique et en Automatique*, n° 114, INRIA, Rocquencourt, 1987, p. 25-29.

[CJJ-83] : *Rapport du sous-comité sur les questions relatives aux ordinateurs*, Comité permanent de la Justice et des questions Juridiques, Canada, juin 1983 (cité dans [HEU-89]).

[COH-85] : COHEN, F., *A Secure Computer Network Design*, in *Computers & Security*, 4(3), 1985, p.189-205.

[COH-87] : COHEN, F., *Computer Viruses : Theory and Experiments*, in *Computers & Security*, 6, 1987, p.22-35.

[COH-88] : COHEN, F., *On the Implication of Computer Viruses and Methods of Defense*, in *Computers & Security*, 7(2), 1988, p.167-184.

[COO-84] : COOK, N., *Protection of Software*, in *Data Processing 26(6)*, juillet/août 1984, p.40-41.

[DAV-84] : DAVIES, D.W., PRICE, W.L., *Security for Computer Networks*, John Wiley & Sons, Chichester, 1984.

[DEN-82] : DENNING, D.E., *Cryptography and Data Security*, Addison Wesley, Reading, MA, 1982.

[DIF-76] : DIFFIE, W., HELLMAN, M., *New Directions in Cryptography*, in *IEEE Trans. Inform. Theory*, IT 22(6), Novembre 1976, p. 644-654.

[ELM-88] : ELMER-DeWITT, P., *Invasion of the Data shatchers !*, in *Time Magazine*, 26 septembre 1988, p.40-45.

[ESS-86] : ESSEN, J., *Security Requirements on Computers and Operating Systems*, in *IFIP/SEC'86*, Monte-Carlo, 1986.

[FAK-88] : FAK, V., *Are we Vulnerable to a Virus Attack : a Report From Sweden*, in *Computers & Security*, 7(2), 1988, p.151-155.

[FER-86] : FERREIRA, R., *Applications and Performances of the Smart Card*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 487-503.

[FIA-87] : FIAT, A., SHAMIR, A., *preuve d'identité infalsifiable*, in *SECURICOM 87*, mars 1987, p. 147-155.

[FIPS-77] : *Data Encryption Standard*, FIPS publication 46, 15/01/1977.

[FIPS-80] : *DES Mode of Operations*, FIPS publication 81, 1980.

[FRA-86] : FRAY, J.M., DESWARTE, Y., POWELL, D., *A Secure Distributed File System Based on Intrusion Tolerance*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 407-416.

[GIL-88] : GILLET, P., *De l'utilisation de fonctions d'identification pour la vérification de l'intégrité des fichiers*, mémoire présenté aux FNDP, Namur, septembre 1988.

[GOL-86] : GOLDREICH, O., MICALI, S., WIGDERSON, A., *Proofs that Yield Nothing but Their Validity*, in *Proceedings of the 27th Annual Symposium of the Foundations of Computer Science*, IEEE Publication, 1986.

[GRA-86] : GRANDJEAN, I., *Computerized Medical Data - Privacy and Deliquency Issues*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 174-184.

[GRU-88] : GRUHIER, F., *Votre ordinateur a la vérole...*, in *Le nouvel Observateur/ Notre époque*, 17-23 novembre 1988, p. 120-123.

[GUI-82] : GUILLOU, L.C., LORIG, B., *Cryptography and Telematics*, in *Computers and Security*, 1(1), Janvier 1982, p. 27-33.

[HER-84] : HERSCHBERG, I.S., PAANS, R., *The Programmer Threat : Cases and Causes*, in *Computers & Security*, 3, 1984, p.263-272.

[HER-87] : HERSCHBERG, I.S., *The Hacker's Comfort*, in *Computers & Security*, 6, 1987, p.133-138.

[HERB-84] : LES HERBST, *Software Protection Using Disk Formatting and Operating System Modifications on MacIntosh/Apple 2/IBM*, 1984.

[HEU-89] : HEUSE, B., LEJOLY, M., ROBAT, D., THERASSE, T., Van ROSSUM, E., *La criminalité informatique*, Travail réalisé dans le cadre du cours d'*Informatique et Société*, BERLEUR, J. (s.j.), FNDP, Namur, 1989.

[HIG-88] : HIGHLAND, H.J., *An Overview of 18 Virus Protection Products*, in *Computers & Security*, 7(2), 1988, p.158-161.

[HOB-84] : Van HOBOKEN, R., *The Burglar's Viewpoint*, in *Computers & Security*, 3, 1984, p.295-302.

[IBM-83] : *RACF General Information Manual (GC28-0722-7)*, IBM publications, New-York, 1983.

[KRA-87] : KRAYEM, R., *La sécurité des systèmes informatiques ouverts et la protection d'accès avec la carte à microprocesseur*, thèse de doctorat de l'ENST, Paris, 1987.

[LAM-88] : LAMERE, J-M., TOURLY, J., *La sécurité des petits et moyens sytèmes informatiques*, DUNOD Informatique, Paris, 1988.

[LER-87] : LEROY, H., *Calculabilité et complexité*, cours dispensé aux FNDP, Namur, 1987.

[LOU-88] : LOUIS, C., *La protection des logiciels : une solution pour ordinateurs personnels utilisant la carte à micro-calculateur*, thèse de doctorat de l'université Pierre et Marie Curie, Paris, Mai 1988.

[MAC-87] : MACCHI, C., GUILBERT, J-F, *Téléinformatique*, Dunod Informatique, Paris, 1987.

[MAD-88] : MADSEN, C.W., *The World Meganetwork and Terrorism*, in *Computers & Security*, 7(4), p.347-352.

[MEI-84] : MEINADIER, J.P., *Structure et fonctionnement des ordinateurs*, Larousse, Paris, 1984.

[MOW-86] : MOWSHOWITZ, *Management, Ethics and Computer Crime*, exposé présenté aux FNDP, FNDP, Namur, 1986.

[NEU-49] : Von NEUMANN, J., *Theory and Organization of Complicated Automata*, 1949.

[PAR-76] : PARKER, D.B., *Crime by Computer*, 1976.

[POZ-86] : POZZO, M., GRAY, T., *Computer Virus Containment in Untrusted Computing Environment*, in *IFIP/SEC'86*, Monte Carlo, 1986, p.118-126.

[POZ-87] : POZZO, M., GRAY, T.E., *An Approach to Containing Computer Viruses*, in *Computers & Security*, 6, 1987, p.321-331.

[RAN-87] : RANEA, P.G., FRAY, J-M, DESWARTE, Y., *SATURNE : un système tolérant les intrusions par la fragmentation-dissémination*, in *Bulletin de liaison de la recherche en Informatique et en Automatique*, n° 114, INRIA, Rocquencourt, 1987, p. 20-24.

[RSA-78] : RIVEST, R.L., SHAMIR, A., ADLEMAN, L., *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, in *Communications of the ACM*, 21(2), Fevrier 1978, p. 120-126.

[SCH-88] : De SCHRYVER, J., *Virus informatiques : décontaminez votre ordinateur*, in *Micro-systèmes*, décembre 1988, p.60-66.

[SHO-82] : SHOCH, J.F., HUPP, J.A., *The "WORM" Programs - Early Experience with a Distributed Computation*, in *Communications of the ACM*, 25(3), Mars 1982, p.172-180.

[THO-84] : THOMSON, K., *Reflections on Trusting Trust*, in *Communications of the ACM*, 27(8), août 1984, p.761-763.

[TUR-50] : TURING, A.M., *Computing Machinery and Intelligence*, in *Mind*, LIX(236), 1950.

[VAX-85] : *Guide to VAX/VMS System Security*, Digital Equipment Corporation, MAYNARD, Ma, Juillet 1985.

[VIT-89] : *L'Ordinateur et après ?*, fiche 11 : *Vulnérabilité et sécurité*, VITAL & Cie, 1989 (cité dans [HEU-89]).

[WAY-87] : WAYNER, P., *Zero Knowledge Proofs*, in *Byte*, octobre 1987, p. 149-152.

BIBLIOGRAPHIE

Anonyme, *Hardware Locks are Key to Keeping Company Data Safe*, in *DEC Computing*, mars 1988.

BAXTER, M.S.J., *A Layered Architecture for Multi-Level Security*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 229-235.

CAMPANA, M., GIRAULT, C., *Comment utiliser les fonctions de condensation dans la protection des données*, in *SECURICOM 88*, mars 1988, P. 91-110.

CARROLL, J.M., *Implementing Multilevel Security by Violation Privilege*, in *Computers & Security*, 7(6), 1988, p. 563-573.

CHAMOUX, J.P., *Menaces sur l'ordinateur*, éditions du Seuil, Paris, 1986.

CHORLEY, B.J., PRICE, W.L., *An Intelligent Token for Secure Transactions*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 442-450.

COHEN, F., *Maintaining a Poor Person's Information Integrity*, in *Computers & Security*, 7(5), 1988, p.489-494.

DMPA code of Ethics and Standards of Conduct for Information Processing Professionals, Viewing the Need, Development, in *Supplement to Data Management*, Mai 1981.

ELOFF, J.H.P., *Computer Security Policy : Important Issues*, in *Computers and Security*, 7, 1988, p. 559-562.

ERKELENS, C., *La délinquance informatique belge et le droit pénal belge*, in *Annexe à Droit de l'Informatique*, 6, 1985, p.21 et suiv.

ESSEN, J., *L'utilisation d'un système de contrôle d'autorisation*, in *SECURICOM 84*, mars 1984.

GLISS, H., *Analysis of the International Hacking Attack Against the NASA SPANet*, in *SECURICOM 88*, Paris, Mars 1988.

GRISSONANCHE, A., *Sécurité : une approche générale*, Congrès Protection et Confidentialité des Systèmes Informatiques, Paris, 14-16 juin 1986.

HERSCHBERG, I.S., *Make the Tigers Hunt for You*, in *Computers & Security*, 7(2), 1988, p.197-207.

HIGHLAND, H.J., *Computer Security : Data Protection Techniques*, in *SECURICOM 84*, mars 1984, p. 119-135.

HIGHLAND, H.J., *Blitzed by a Computer Virus*, in *Computers & Security*, 7(2), 1988, p.237-239.

HIGHLAND, H.J., *Virus Defense Alert*, in *Computers & Security*, 7(2), 1988, p.156-158.

ILIE, F., *La sécurité informatique avec la carte à micro-calculateur*, these de doctorat de l'Ecole Nationale Supérieure des Télécommunications, Paris, Mai 1988.

JOLIA-FERRIER, L., *Penetration Testing as an Audit Tool*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 140-145.

JONES, R.W., *The Creation and Use of Explicit Rights in a Distributed System*, in *IFIP/SEC'86*, Monte-Carlo, 1986, p. 374-385.

KINNE, D., *Comment réaliser la protection de vos ressources sur micro-ordinateur*, in *SECURICOM 84*, 1984.

LAMERE, J.M., *La sécurité informatique, approche méthodologique*, Dunod, Paris, 1986.

MAHL, R., *Parades et attaques possibles contre la sécurité d'UNIX*, in *UNIX usr/info*, 4(2), Février 1988.

MARIN, B., *Secur'Acces : système de contrôle d'accès par carte CP8*, in *SECURICOM 87*, mars 1987, p. 187-196.

MAUDE, T., MAUDE, D., *Hardware Protection Against Software Piracy*, in *Communications of the ACM*, 27(9), septembre 1984, p.950-959.

MEYER, C., SHILLING, M., *Chargement sécurisé d'un programme avec code de détection de manipulation*, in *SECURICOM 88*, mars 1988, p. 111-130.

MORRIS, R., THOMPSON, K., *Password Security : A Case History*, in *Communications of the ACM*, 22(11), novembre 1979, p. 594-597.

MURRAY, W.H., *The Application of Epidemiology to Computer Viruses*, in *Computers & Security*, 7(2), 1988, p.139-150.

PUJOLLE, G., SERET, D., DROMARD, D., HORLAIT, E., *Réseaux et télématique*, tomes 1 & 2, Eyrolles, Paris, 1985.

RICHY, P., *Le point sur les différentes classes de sécurisation*, in *UNIX usr/info*, 4(4), Avril 1988.

SHANNON, C.E., *Communication Theory of Secrecy Systems*, in *Bell Technical Journal*, 1949, p.28.

SIEBER, U., *The International Handbook on Computer Crime*, John Wiley & sons, New-York , 1986.

SPREUTELS, *Infractions liées à l'informatique en droit belge*, in *Revue de droit pénal et de criminologie*, 1985, p.357 et suiv.

The British Computer Society, *Code of Conduct*, Handbook 5, octobre 1983.

The British Computer Society, *Code of Practice*, Handbook 6, Septembre 1983.

WERNERY, S., *The Chaos Computer Club and Hacking*, in *SECURICOM 88*, Paris, mars 1988.